Technical Report

IRI-TR-19-02

# A Randomized Kinodynamic Planner for Closed-chain Robotic Systems

R. Bordalba
L. Ros
J. M. Porta

Institut de Robòtica i Informàtica Industrial

## Abstract

Kinodynamic RRT planners are effective tools for finding feasible trajectories in many classes of robotic systems. However, they are hard to apply to systems with closed-kinematic chains, like parallel robots, cooperating arms manipulating an object, or legged robots keeping their feet in contact with the environment. The state space of such systems is an implicitly-defined manifold, which complicates the design of the sampling and steering procedures, and leads to trajectories that drift away from the manifold when standard integration methods are used. To address these issues, this report presents a kinodynamic RRT planner that constructs an atlas of the state space incrementally, and uses this atlas to both generate random states, and to dynamically steer the system towards such states. The steering method is based on computing linear quadratic regulators from the atlas charts, which greatly increases the planner efficiency in comparison to the standard method that simulates random actions. The atlas also allows the integration of the equations of motion as a differential equation on the state space manifold, which eliminates any drift from such manifold and thus results in accurate trajectories. To the best of our knowledge, this is the first kinodynamic planner that explicitly takes closed kinematic chains into account. We illustrate the performance of the approach in significantly complex tasks, including planar and spatial robots that have to lift or throw a load at a given velocity using torque-limited actuators.

**Institut de Robòtica i Informàtica Industrial (IRI)**
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain
Tel (fax): +34 93 401 5750 (5751)
http://www.iri.upc.edu

**Corresponding author:**
R. Bordalba
tel: +34 93 4015806
rbordalba@iri.upc.edu

# 1 Introduction

Since its formalization in the early nineties [21], the kinodynamic planning problem remains 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 system. The problem is difficult to solve in general. To ensure feasibility, the trajectory should: 1) fulfil all kinematic constraints of the system, including holonomic ones, like loop-closure or end-effector constraints, or nonholonomic ones, like rolling contact or velocity limit constraints; 2) be compliant with the equations of motion of the robot; 3) avoid the collisions with static or moving obstacles in the environment; and 4) be executable with the limited force capacity of the actuators. In certain applications, moreover, the trajectory should also be optimal in some sense, minimising, for example, the time or control effort required for its execution.

The ability to plan such trajectories is key in a robotic system. Above all, it endows the system with a means to convert higher-level commands—like "move to a certain location", or "throw the object at a given speed"—into appropriate reference signals that can be followed by the actuators. By accounting for the robot dynamics and force limitations at the planning stage, moreover, the motions are easier to control, and they often look more graceful, or physically natural [43], as they tend to adapt to the normal modes of oscillation of the system, taking advantage of gravity, inertia, and centripetal forces to the benefit of the task.

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 its obstacle region is virtually larger, involving those 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 between nearby states is considerably more difficult as well. While straight-line motions usually suffice in configuration space, steering motions in the state space must conform to the vector fields defined by the equations of motion and the actuator limits of the robot.

Among all kinodynamic planning techniques, the rapidly-exploring random tree (RRT) method has emerged as one of the most successful algorithms to date [42]. This algorithm makes intensive use of sampling and dynamic simulations to grow trajectory trees over the state space until the start and goal states get connected. The efficiency of the approach is remarkable, especially in view of its simplicity and relative ease of implementation. The method is fairly general and, with proper extensions, can even converge to minimum-cost motions [31, 45]. However, the algorithm also suffers from an important limitation: it assumes that the robot state can be described by means of independent generalised coordinates. This implies that the algorithm is directly applicable to open-chain robots, or to robots with explicit state space parametrisations, but it has problems in dealing with general mechanisms involving closed-kinematic chains. Such chains arise frequently in today's robots and manipulation systems (Fig. 1), which explains the growing interest they arouse in the literature [2, 30, 34, 37, 38, 53, 54].

Unlike in the open-chain case, the state space of a closed-chain robot is not flat anymore. Instead, it is a nonlinear manifold defined implicitly by a system of equations that, in general, cannot be solved in closed form. This manifold is a zero-measure set in a larger ambient space, which complicates the design of sampling and steering methods to explore the manifold efficiently. Moreover, if the dynamic model of the robot is not properly handled, the state space trajectories may deviate substantially from the manifold, leading to undesired violations of the kinematic constraints. Forward singularities in which the robot is locally underactuated also complicate the planning and control of motions across certain surfaces of the state space [10].

The purpose of this report is to extend the planner in [42] to cope with the previous complications. As we shall see, by constructing an atlas of the state space in parallel to the RRT, one can define proper sampling and steering methods that deal with closed kinematic chains effectively, while producing accurate dynamic simulations of the system even across forward singularities. A preliminary

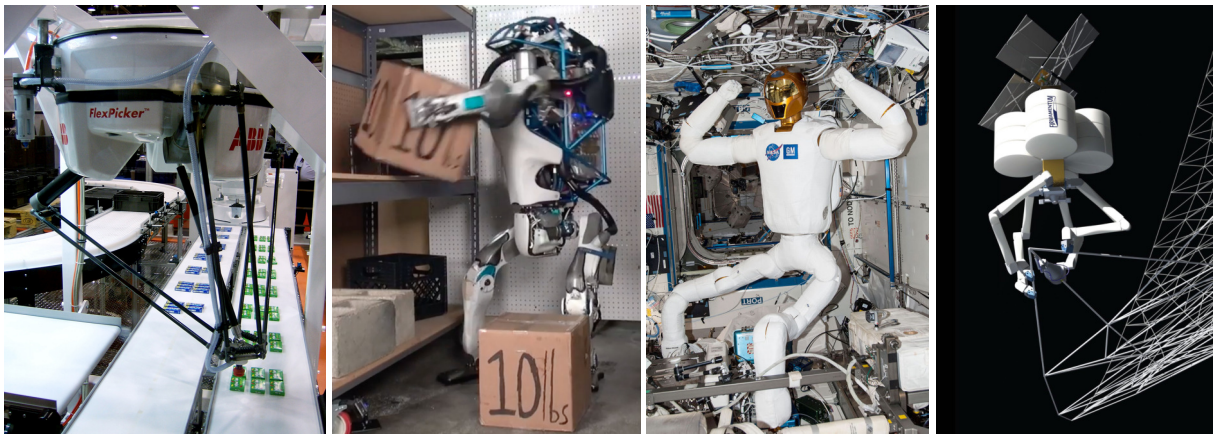



Figure 1: Example systems involving closed kinematic chains. The chains may be intrinsic to the robot structure, as in parallel robots (left picture), or they may result from contact constraints during a task, as in multi-limb systems manipulating an object, or keeping legs in contact with the environment (right pictures). From left to right: A Delta parallel robot [11], the Atlas robot from Boston Dynamics lifting a heavy load [26], the Robonaut 2 robot with two legs clamped to the International Space Station [20], and the SpiderFab Bot, a conceptual design for self-fabricating space systems [33]. Pictures courtesy of ABB, Boston Dynamics, NASA, and Tethers Unlimited, Inc (respectively).

version of the planner we propose was presented in [14]. In comparison to [14], this report develops a new steering method based on linear quadratic regulators (LQR), which greatly increases the efficiency of the planner in comparison to the randomized steering strategy in [14]. New challenging test cases are also included for demonstration, including tasks that require the throwing of objects at a given velocity, and bimanual manipulations of heavy loads, which were difficult to solve with [14]. It is worth noting that, while some path planning approaches have previously dealt with closed kinematic chains [2, 18, 29, 34, 37, 54, 65, 70], none of these approaches has considered the dynamics of the system into the planner. Our kinodynamic planner, in fact, can also be seen as an extension of the work in [34] to cope with dynamic constraints.

The rest of this document is organized as follows. Section 2 briefly reviews the state of the art on kinodynamic planning to better place our work into context. Section 3 formally states the problem we confront, taking into account the various constraints intervening. Section 4 recalls the standard RRT method in [42] and describes, at the same time, the main difficulties it exhibits in the presence of closed kinematic chains. Sections 5 and 6 then present effective sampling, simulation, and steering methods that allow overcoming such difficulties. The resulting tools are used in Section 7 to implement the planning method we propose, whose performance is examined in Section 8 using illustrative examples. Section 9 finally provides the conclusions and discusses several points deserving further attention. In the end of the we also include three appendices that prove some needed results, and describe the constraint formulations that we adopt in the planner.

# 2 Related work

## 2.1 Configuration space approaches

The sheer complexity of kinodynamic planning is usually managed by decomposing the problem into two simpler problems [41]. Initially, the dynamic constraints of the robot are neglected and a collision-free path in the C-space is sought that solely satisfies the kinematic constraints. Then, a time-parametric trajectory constrained to the previous path is designed while accounting for the dynamic constraints and

force limits of the actuators. Although many techniques can be used to compute the path, including probabilistic roadmaps or randomized tree techniques among others [17, 41], the trajectory is usually obtained with the time-scaling method in [8], or its later improvements [49, 62–64]. This method regards 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 finds a monotonic time scaling $s = s(t)$ such that $\boldsymbol{q}(t) = \boldsymbol{q}(s(t))$ connects the start and goal configurations in minimum time. The method is fast and elegant, as it exploits the bang-bang nature of the solution in the $(s, \dot{s})$ plane, and robust implementations have recently been developed [52].

The previous approach obtains a trajectory that is only time-optimal for the computed path, but it is appealing in that it makes the problem more tractable. The approach has been the method of choice traditionally, as it proves to be effective in systems with many degrees of freedom like humanoids, legged robots, or mobile robot formations [50]. Its lack of completeness, moreover, can always be alleviated by improving the trajectory a posteriori, using appropriate optimization techniques [7, 60, 61]. 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 can be combined with randomized planners to generate graceful dynamic motions [50].

It must be noted that, 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 cannot generate swinging paths in principle, and such paths may be required in highly dynamic tasks like lifting a heavy load under strict torque limitations. In other approaches, start or goal states with nonzero velocity cannot be specified, which is necessary in catching or throwing objects at a certain speed and direction. Time scaling methods, moreover, require the robot to be fully actuated. While this is rarely an issue in robot arms or humanoids under contact constraints [30, 53], parallel robots with passive joints make the robot underactuated at forward singularities [10]. These configurations are problematic when managed in the C-space as they can only be traversed under particular velocities and accelerations. As we shall see in this report, on the contrary, the previous limitations do not apply when robot trajectories are directly planned in the state space.

## 2.2 State space approaches

Existing techniques for planning in the state space can roughly be grouped into optimization and randomized approaches. On the one hand, optimization approaches can be applied to remarkably-complex problems [4, 35, 56, 59, 71]. An advantage is that they can accommodate a wide variety of kinematic and dynamic constraints. For instance, differential constraints describing the robot dynamics can be enforced by discretising the trajectory into different knot points using an Euler method, or any higher-order method if more accuracy is necessary. However, there is a trade-off between the number of knot points and integration method adopted, and the computational cost of the resulting optimization problem. In systems with closed kinematic chains, moreover, the discretization of the differential equations produces knot points that easily drift away from the state space manifold, which results in unwanted link disassemblies, or contact losses. In [56], differential constraints were approximated explicitly by means of polynomial functions while guaranteeing third-order integration accuracy in constrained systems. Nevertheless, the problem size becomes huge for long time horizons or systems with many degrees of freedom [50]. Good discussions on the advantages and pitfalls of optimization-based techniques can be found in [5] and [30]. On the other hand, randomized approaches like the RRT can cope with differential constraints in relatively high-dimensional problems, and guarantee to find a feasible solution when it exists and enough computing time is available [42]. Their main issue, however, is that exact steering methods are not available for nonlinear dynamical systems. The standard RRT method tries to circumvent this issue by simulating random actions for a given time, and then selecting the action that gets the system closest to the target [42]. For particular systems, better solutions exist. For instance, the approach in [40] assumes double integrator dynamics for the systems, and then 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 linearisation, which might require unfeasible torques from the motors. The method in [47] linearises the system dynamics and uses an infinite-horizon LQR controller to define a steering method, but such a controller cannot be used to reach goal states with nonzero velocity. In contrast, [27] and [69] use finite-horizon LQR controllers that yield open-loop control policies. The LQR-trees algorithm in [66] uses the same control policy to initialize a trajectory optimization process with the full nonlinear dynamics of the system. As designed, however, the previous steering methods cannot be applied to robots with closed kinematic chains, as they assume the state coordinates to be independent. Our steering approach is similar to those in [27, 66, 69], but extended to also cope with closed kinematic chains.

## 3 Problem formulation

To formulate our problem, we describe the robot configuration by means of a tuple $\boldsymbol{q}$ of $n_q$ generalized coordinates, which determine the positions and orientations of all links at a given instant of time. We restrict our attention to robots with closed kinematic chains, in which $\boldsymbol{q}$ must satisfy a system of $n_e$ nonlinear equations

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \mathbf{0} \tag{1}$$

enforcing the closure conditions of the chains. The C-space of the robot is then the set

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

whose structure may be quite complex in general. For simplicity, however, we assume that the Jacobian $\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q}) = \partial\boldsymbol{\Phi}/\partial\boldsymbol{q}$ is full rank for all $\boldsymbol{q} \in \mathcal{C}$, so that $\mathcal{C}$ is a smooth manifold of dimension $d_{\mathcal{C}} = n_q - n_e$. This assumption is not too restrictive, as geometric singularities can be removed from $\mathcal{C}$ by judicious mechanical design [10], or through the addition of singularity-avoidance constraints [9, 12]. By differentiating Eq. (1) with respect to time, we also obtain the velocity equation of the robot

$$\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}} = \mathbf{0}, \tag{2}$$

which characterises the feasible vectors $\dot{\boldsymbol{q}}$ at a given $\boldsymbol{q} \in \mathcal{C}$. Appendix B explains the particular formulations of Eqs. (1) and (2) that we adopt in our implementation.

Let $\boldsymbol{F}(\boldsymbol{x}) = \mathbf{0}$ denote the system formed by Eqs. (1) and (2), where $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n_x}$ is the state vector of the robot, with $n_x = 2n_q$. While path planning approaches operate in $\mathcal{C}$, kinodynamic planning problems are better represented in the state space

$$\mathcal{X} = \{\boldsymbol{x} : \boldsymbol{F}(\boldsymbol{x}) = \mathbf{0}\}. \tag{3}$$

It can be shown that, since $\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})$ is full rank in our case, $\mathcal{X}$ is also a smooth manifold, but of dimension $d_{\mathcal{X}} = 2\, d_{\mathcal{C}}$. This implies that the tangent space of $\mathcal{X}$ at $\boldsymbol{x}$,

$$\mathcal{T}_{\boldsymbol{x}}\mathcal{X} = \{\dot{\boldsymbol{x}} \in \mathbb{R}^{n_x} : \boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x})\, \dot{\boldsymbol{x}} = \mathbf{0}\}, \tag{4}$$

is well-defined and $d_{\mathcal{X}}$-dimensional for any $\boldsymbol{x} \in \mathcal{X}$.

We shall encode the forces and torques of the actuators into an action vector $\boldsymbol{u} = (u_1, \ldots, u_{n_u}) \in \mathbb{R}^{n_u}$. Given a starting state $\boldsymbol{x}_s \in \mathcal{X}$, and the vector $\boldsymbol{u}$ as a function of time, $\boldsymbol{u} = \boldsymbol{u}(t)$, the time evolution of the robot is determined by a system of differential-algebraic equations of the form

$$\begin{cases} \boldsymbol{F}(\boldsymbol{x}) = \mathbf{0}, & (5) \\ \dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}). & (6) \end{cases}$$

In this system, Eq. (5) forces the states $\boldsymbol{x}$ to remain in $\mathcal{X}$, and Eq. (6) models the dynamics of the robot. As explained in Appendix C, Eq. (6) can be obtained from the multiplier form of the Euler-Lagrange

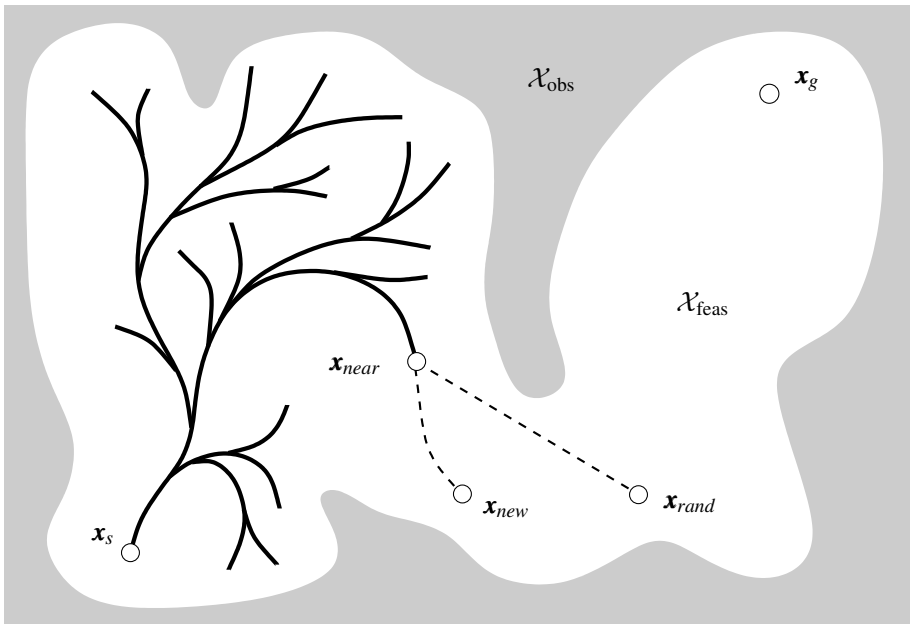



Figure 2: The standard RRT method in [42].

equations for example. Note that for each value of $\boldsymbol{u}$, Eq. (6) defines a vector field over $\mathcal{X}$, which can be used together with Eq. (5) to integrate the robot motion forward in time, using proper numerical methods [57].

To model the fact that the actuator forces are limited, we will assume that $\boldsymbol{u}$ can only take values inside the box

$$\mathcal{U} = [-l_1, l_1] \times [-l_2, l_2] \times \ldots \times [-l_{n_u}, l_{n_u}] \tag{7}$$

of $\mathbb{R}^{n_u}$, where $l_i$ denotes the limit force or torque that the $i$-th motor can exert. For simplicity, the $l_i$ values are taken to be constant, but our algorithms could also be adapted to cope with state-dependent bounds if desired. Along a trajectory, moreover, the robot cannot incur in collisions with itself or with the environment, and should fulfil any limits imposed on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. This reduces the feasible states $\boldsymbol{x}$ to those lying in a subset $\mathcal{X}_{\text{feas}} \subseteq \mathcal{X}$.

With the previous definitions, the planning problem we confront can be phrased as follows: Given a kinematic and dynamic model of the 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 policy $\boldsymbol{u} = \boldsymbol{u}(t) \in \mathcal{U}$ such that the trajectory $\boldsymbol{x} = \boldsymbol{x}(t)$ determined by Eqs. (5) and (6) for $\boldsymbol{x}(0) = \boldsymbol{x}_s$ fulfils $\boldsymbol{x}(t_f) = \boldsymbol{x}_g$ for some time $t_f > 0$, with $\boldsymbol{x}(t) \in \mathcal{X}_{\text{feas}}$ for all $t \in [0, t_f]$.

Observe that, in contrast to [42], we allow the presence of Eq. (5) in our planning problem, which makes it more general and challenging at the same time. Thus, whereas in [42] $\mathcal{X}$ is simply $\mathbb{R}^{n_x}$, in our case $\mathcal{X}$ is a lower dimensional manifold embedded in $\mathbb{R}^{n_x}$. In [41], minor modifications to [42] were suggested to cope with such manifolds, but we next explain that these lead to several complications.

## 4 Drawbacks of the standard RRT method

Recall from [42] that a standard RRT is initialized at $\boldsymbol{x}_s$ and it is expanded by applying the following steps repeatedly (see Fig. 2): 1) a guiding state $\boldsymbol{x}_{rand} \in \mathcal{X}$ is randomly selected; 2) the RRT state $\boldsymbol{x}_{near}$ that is closest to $\boldsymbol{x}_{rand}$ 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 $\boldsymbol{x}_{rand}$ in the absence of obstacles; and 4) the movement that results from applying $\boldsymbol{u}$ during some time $\Delta t$ is obtained by integrating Eq. (6). 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

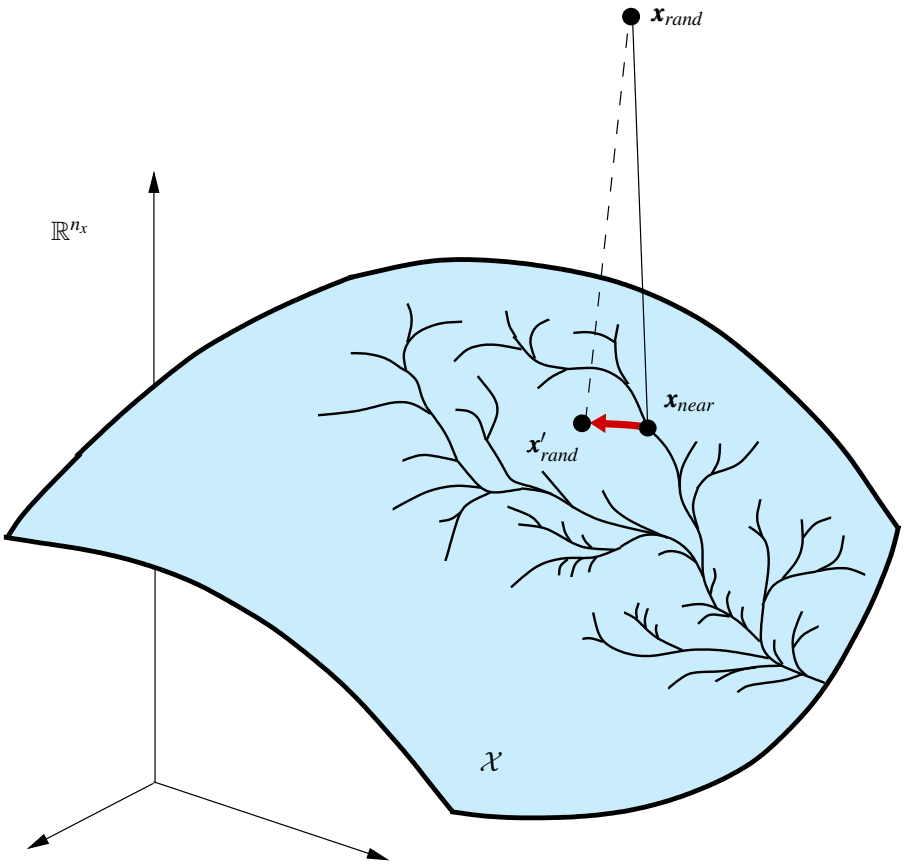



Figure 3: Generation of a guiding sample according to [41].

former case, the action $\boldsymbol{u}$ is stored in the new edge connecting $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{new}$. The process stops 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 $\boldsymbol{x}_{rand}$.

Three complications arise when applying the previous method to closed kinematic chains. First, the points $\boldsymbol{x}_{rand}$ are difficult to obtain in general, as $\mathcal{X}$ will not admit explicit parametrizations. To avoid this problem, [41, Sec. 7.4.1] proposes to randomly pick $\boldsymbol{x}_{rand}$ from the larger ambient space $\mathbb{R}^{n_x}$ (Fig. 3) and use, as a guiding state, the point $\boldsymbol{x}'_{rand}$ that results from projecting $\boldsymbol{x}_{rand}$ onto the tangent space of $\mathcal{X}$ at $\boldsymbol{x}_{near}$. However, while $\boldsymbol{x}'_{rand}$ is easy to compute, its pulling effect on the RRT may be small. Notice that the ambient space could be large in comparison to $\mathcal{X}$, resulting in points $\boldsymbol{x}'_{rand}$ that might often be close to $\boldsymbol{x}_{near}$, which diminishes the Voronoi bias of the RRT. All these effects were analysed in [34] and [36]. A second complication concerns the dynamic simulation of robot motions. The standard RRT method would only use Eq. (6) to generate such motions, on the grounds that Eq. (5) is accounted for implicitly by Eq. (6) [41, Sec. 13.4.3.1]. However, from multibody mechanics it is known that the motion of a closed-chain mechanism can only be predicted reliably if Eq. (5) is actively used during the integration of Eq. (6) [48]. Otherwise, the inevitable errors introduced when discretising Eq. (6) will make the trajectory $\boldsymbol{x}(t)$ increasingly drift away from $\mathcal{X}$ as the simulation progresses. It is shown in [14] that such a drift may even be large enough to prevent the connection of $\boldsymbol{x}_s$ with $\boldsymbol{x}_g$. The use of Baumgarte stabilization to compensate this drift [25] is also problematic, as it may lead to instabilities [6], it alters the system energy artificially, and its stabilising parameters are not easy to tune in general. A third difficulty, finally, concerns the steering method. A simple strategy based on simulating random actions from $\mathcal{U}$ is proposed in [42]. This strategy is easy to implement, but it can be inefficient because the number of samples required 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. In fact, existing methods to alleviate this problem have only been given for open-chain robots to date [27, 40, 47, 66, 69].

In the next two sections we shall see that the previous difficulties can all be overcome by constructing an atlas of $\mathcal{X}$. The atlas will provide us with the tools to: 1) sample the $\mathcal{X}$ manifold directly instead of its ambient space $\mathbb{R}^{n_\mathcal{X}}$; 2) integrate Eqs. (5) and (6) as a true differential-algebraic equation to guarantee driftless motions on $\mathcal{X}$; and 3) define an effective steering method for closed kinematic chains.

# 5 Mapping and exploring the state space

## 5.1 Atlas construction

Formally, an atlas of $\mathcal{X}$ is a collection of charts mapping $\mathcal{X}$ entirely, where each chart $c$ is a local diffeomorphism $\boldsymbol{\varphi}_c$ from an open set $V_c \subset \mathcal{X}$ to an open set $P_c \subseteq \mathbb{R}^{d_\mathcal{X}}$ [Fig. 4(a)]. 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 $\boldsymbol{y} = \boldsymbol{\varphi}_c(\boldsymbol{x})$ provides the local coordinates, or parameters, of $\boldsymbol{x}$ in chart $c$. Since each map $\boldsymbol{\varphi}_c$ is a diffeomorphism, its inverse map $\boldsymbol{\psi}_c = \boldsymbol{\varphi}_c^{-1}$ also exists, and gives a local parametrisation of $V_c$.

For particular manifolds, $\boldsymbol{\varphi}_c$ and $\boldsymbol{\psi}_c$ can be defined in closed form. However, we propose to use the tangent space parametrization [32] to define them for any manifold. Using this parametrisation, the map $\boldsymbol{y} = \boldsymbol{\varphi}_c(\boldsymbol{x})$ 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. 4(b)], so this map takes the form

$$\boldsymbol{y} = \boldsymbol{U}_c^\top (\boldsymbol{x} - \boldsymbol{x}_c), \tag{8}$$

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

$$\left.\begin{array}{c} \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0} \\ \boldsymbol{U}_c^\top (\boldsymbol{x} - \boldsymbol{x}_c) - \boldsymbol{y} = \boldsymbol{0} \end{array}\right\} \tag{9}$$

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

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. (6) into one in the coordinate spaces $P_c$. The time derivative of Eq. (8), $\dot{\boldsymbol{y}} = \boldsymbol{U}_c^\top \dot{\boldsymbol{x}}$, gives the relationship between the two vector fields, and allows writing

$$\dot{\boldsymbol{y}} = \boldsymbol{U}_c^\top \boldsymbol{g}(\boldsymbol{\psi}_c(\boldsymbol{y}), \boldsymbol{u}) = \tilde{\boldsymbol{g}}(\boldsymbol{y}, \boldsymbol{u}), \tag{10}$$

which is Eq. (6), 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 differential-algebraic equations as ordinary differential equations on manifolds [28]. 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. (10), and finally getting $\boldsymbol{x}_{k+1} = \boldsymbol{\psi}_c(\boldsymbol{y}_{k+1})$. 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. (5).

## 5.2 Incremental atlas and RRT expansion

One could construct a full atlas of the implicitly-defined state space and then use its local parametrisations to implement a kinodynamic RRT planner. 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 given motion planning problem. As an alternative, we thus propose to combine the construction of the atlas and the expansion of the RRT [34]. In this approach, a partial atlas is used to both generate random states

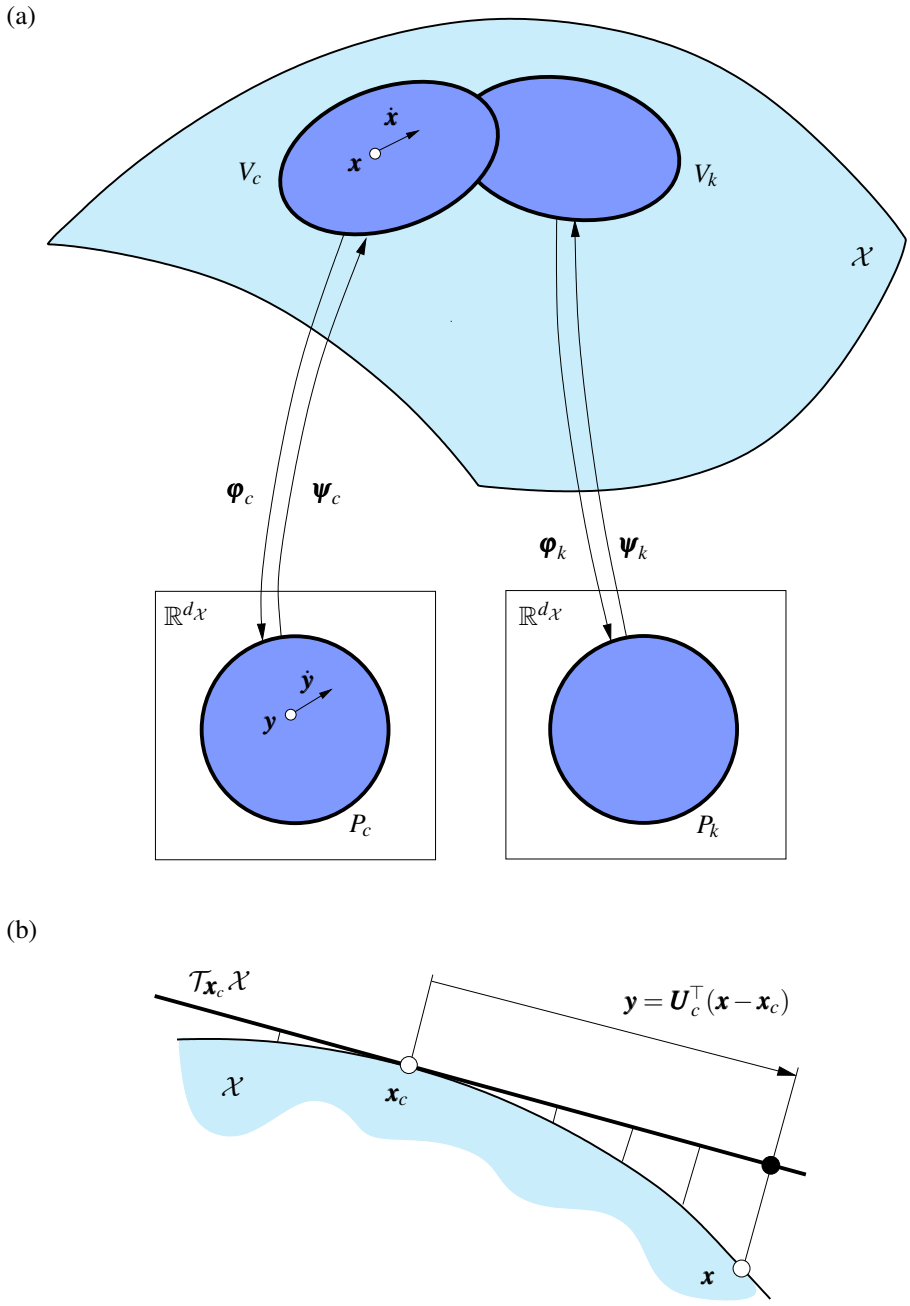



Figure 4: (a) Two neighbouring charts of $\mathcal{X}$, labelled $c$ and $k$, together with their maps $\varphi_c$ and $\varphi_k$, and inverse maps $\psi_c$ and $\psi_k$. (b) Using the tangent space parametrisation, $\varphi_c$ is defined by the projection of $x$ onto $\mathcal{T}_{x_c}\mathcal{X}$.

and grow the RRT branches. As described next, new charts are also created as the RRT branches reach unexplored regions of the state space.

Suppose that $x_k$ and $x_{k+1}$ are two consecutive states along an RRT branch, both covered by a chart at $x_c$, and let $y_k$ and $y_{k+1}$ be their local coordinate vectors in $\mathcal{T}_{x_c}\mathcal{X}$. Then, a new chart at $x_k$ is created if Eq. (9) cannot be solved for $x_{k+1}$ using the Newton-Raphson method, or if any of the following conditions

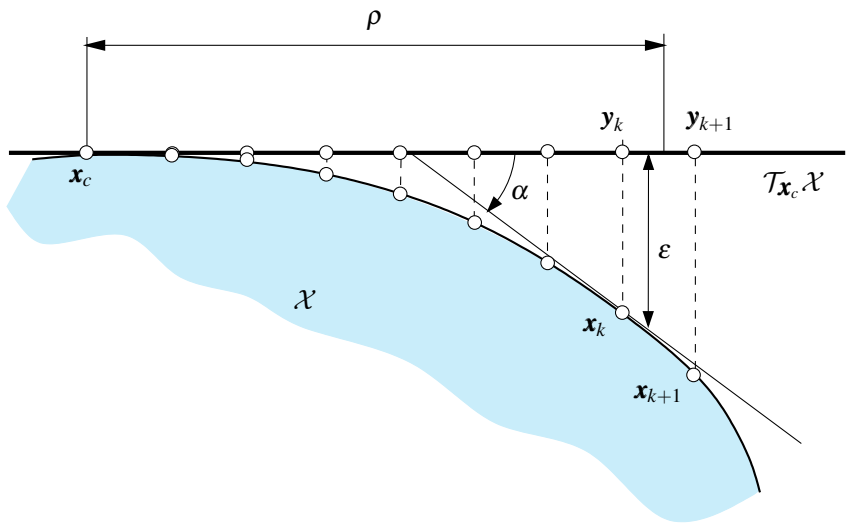



Figure 5: Thresholds determining the extension of the $P_c$ set of the chart at $\boldsymbol{x}_c$. While $\boldsymbol{y}_k$ lies in $P_c$, $\boldsymbol{y}_{k+1}$ does not because it violates Eqs. (11)-(13).

is met

$$\|\boldsymbol{x}_{k+1} - (\boldsymbol{x}_c + \boldsymbol{U}_c\, \boldsymbol{y}_{k+1})\| > \varepsilon, \tag{11}$$

$$\frac{\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\|}{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|} < \cos\alpha, \tag{12}$$

$$\|\boldsymbol{y}_{k+1}\| > \rho, \tag{13}$$

where $\varepsilon$, $\alpha$, and $\rho$ are user-defined thresholds (Fig. 5). 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 the tangent space 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 neighbouring charts. The third condition finally guarantees the generation of new charts as the RRT grows, even for almost flat manifolds.

## 5.3 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 so, the $P_c$ set of each chart is initialized as a ball of radius $\sigma$ centred at the origin of $\mathbb{R}^{d_\mathcal{X}}$. This ball is progressively bounded as new neighbouring charts are created around the chart. If, while growing an RRT branch, a neighbouring chart is created at a point $\boldsymbol{x}_k$ with parameter vector $\boldsymbol{y}_k$ in $P_c$, the following inequality

$$\boldsymbol{y}^\top \boldsymbol{y}_k - \frac{\|\boldsymbol{y}_k\|^2}{2} \leq 0 \tag{14}$$

is added as a bounding half-plane of $P_c$ (Fig. 6). An analogous inequality is added to the $P_k$ set of the chart at $\boldsymbol{x}_k$, but using $\boldsymbol{y}_c = \boldsymbol{\varphi}_k(\boldsymbol{x}_c)$ instead of $\boldsymbol{y}_k$ in Eq. (14). Note that the radius $\sigma$ of the initial ball must be larger than $\rho$ to guarantee that the RRT branches covered by chart $c$ will eventually trigger the generation of new charts, i.e., to guarantee that Eq. (13) will eventually hold. Also, since Eq. (13) forces the norm of $\boldsymbol{y}_k$ to be limited by $\rho$, the half-plane defined by Eq. (14) will be guaranteed to clip $P_c$. Consequently, the $P_c$ sets of those charts fully surrounded by neighbouring charts will be significantly

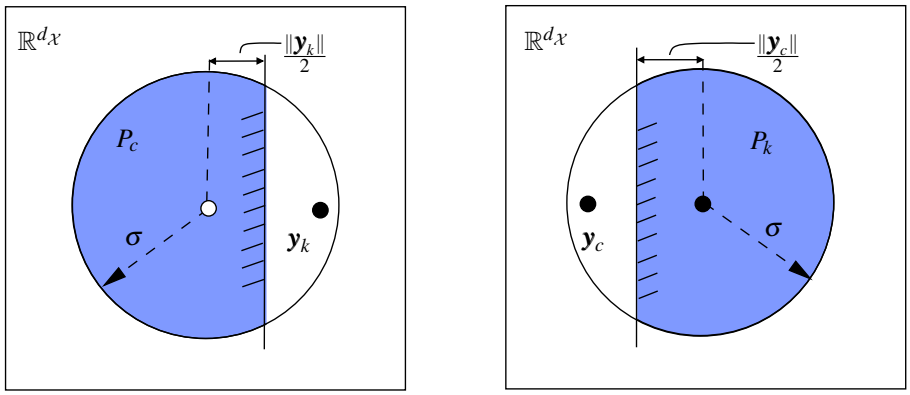



Figure 6: Half planes added to trim the $P_c$ and $P_k$ sets of two neighboring charts. Note that $\boldsymbol{y}_k = \boldsymbol{\varphi}_c(\boldsymbol{x}_k)$ and $\boldsymbol{y}_c = \boldsymbol{\varphi}_k(\boldsymbol{x}_c)$.

smaller than the $P_c$ sets of the charts at the exploration border of the atlas. As we shall see below, this will favour the growth of the tree towards unexplored regions of $\mathcal{X}$.

# 6 A steering method

As explained in Sec. 4, the standard RRT algorithm relies on a randomized steering method that is inefficient when $n_u$ is large. To address this problem, we here propose an alternative approach based on linear quadratic regulators. As we shall see, by linearising the system dynamics at the various chart centres we will be able to obtain a sequence of control laws bringing the robot from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{rand}$.

## 6.1 System linearisation at a chart centre

To apply LQR techniques to our problem we must first linearise our system model at the chart centres $\boldsymbol{x}_c$ and null action $\boldsymbol{u} = \boldsymbol{0}$. To do so, we cannot linearise Eq. (6) however, as this would disregard the fact that the $\boldsymbol{x}$ variables are coupled by Eq. (5). We must instead linearise Eq. (10), which expresses Eq. (6) in the independent $\boldsymbol{y}$ coordinates of $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$. Since the point $\boldsymbol{x} = \boldsymbol{x}_c$ corresponds to $\boldsymbol{y} = \boldsymbol{0}$ in the local coordinates of $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$, the sought linearisation is

$$\dot{\boldsymbol{y}} = \underbrace{\left.\frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{y}}\right|_{\substack{\boldsymbol{y}=\boldsymbol{0}\\ \boldsymbol{u}=\boldsymbol{0}}}}_{\boldsymbol{A}} \boldsymbol{y} + \underbrace{\left.\frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{y}=\boldsymbol{0}\\ \boldsymbol{u}=\boldsymbol{0}}}}_{\boldsymbol{B}} \boldsymbol{u} + \underbrace{\tilde{\boldsymbol{g}}(\boldsymbol{0},\boldsymbol{0})}_{\boldsymbol{c}}, \tag{15}$$

which can be written as

$$\dot{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{c}. \tag{16}$$

This system will be assumed to be controllable hereafter.

Observe that, in Eq. (16), 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, because $(\boldsymbol{x},\boldsymbol{u}) = (\boldsymbol{x}_c,\boldsymbol{0})$ is not necessarily an equilibrium point of the system in Eq. (10). Moreover, by applying the chain rule and using the fact that $\left.\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{y}}\right|_{\boldsymbol{y}=\boldsymbol{0}} = \boldsymbol{U}_c$ (see Appendix A), the $\boldsymbol{A}$ and $\boldsymbol{B}$ terms can be written as:

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

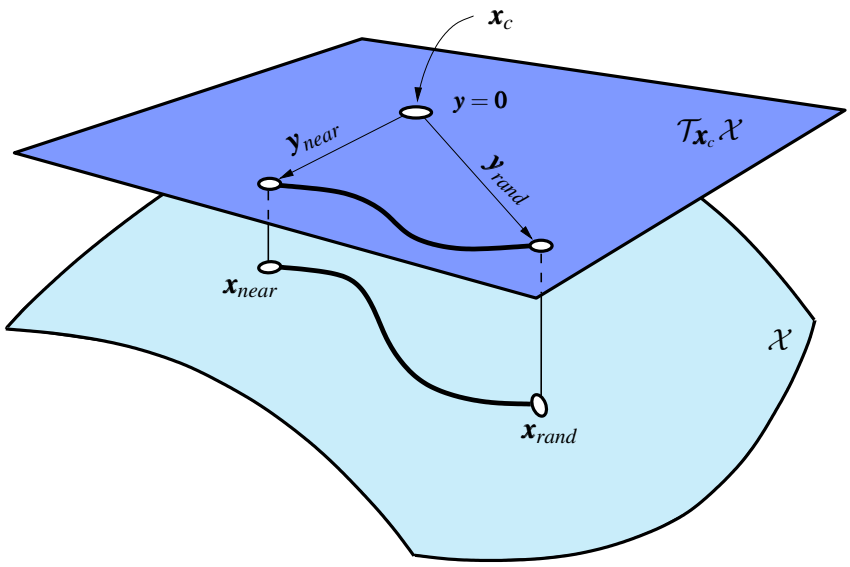



Figure 7: When $x_{near}$ and $x_{rand}$ are covered by a same chart, the steering of the system can be reduced to a steering problem in $\mathcal{T}_{x_c}\mathcal{X}$.

and

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

Notice, therefore, that $A$, $B$, and $c$ can exactly be obtained by evaluating the original function $g(x,u)$ and its derivatives $\partial g/\partial x$ and $\partial g/\partial u$ at $(x,u) = (x_c, 0)$. In those robots in which these derivatives are not easy to obtain in closed form, $A$ and $B$ can always be approximated numerically using finite differences.

## 6.2 Steering on a single chart

Suppose now that both $x_{near}$ and $x_{rand}$ lie in the same chart $c$, centred at $x_c \in \mathcal{X}$ (Fig. 7). In this case, the problem of steering the robot from $x_{near}$ to $x_{rand}$ can be reduced to that of steering the system in Eq. (16) from $y_{near} = \varphi_c(x_{near})$ to $y_{rand} = \varphi_c(x_{rand})$. This problem can be formulated as follows: Find the control policy $u(t) = u^*(t)$ and time $t_f = t_f^*$ that minimize the cost function

$$J(u(t), t_f) = \int_0^{t_f} \left(1 + u(t)^\top R\, u(t)\right) \mathrm{d}t, \tag{17}$$

subject to the constraints

$$\dot{y} = Ay + Bu + c, \tag{18}$$
$$y(0) = y_{near}, \tag{19}$$
$$y(t_f) = y_{rand}. \tag{20}$$

In Eq. (17), the unit term inside the integral penalizes large values of $t_f$, while the term $u(t)^\top R\, u(t)$ penalizes high control actions. In this term, $R$ is a symmetric positive-definite matrix that is known beforehand.

The problem just formulated is known as the fixed final state optimal control problem [44]. We shall solve this problem in two stages. Initially, we will obtain $u^*(t)$ assuming that $t_f$ is fixed, and then we will find a time $t_f$ that leads to a minimum of $J(u(t), t_f)$.

### 6.3 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 [3]. To apply the principle, we first define the Hamiltonian function

$$H(\boldsymbol{y},\boldsymbol{u},\boldsymbol{\lambda}) = 1 + \boldsymbol{u}^\top\,\boldsymbol{R}\,\boldsymbol{u} + \boldsymbol{\lambda}^\top\,(\boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{c})\,, \tag{21}$$

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

$$\dot{\boldsymbol{y}} = \frac{\partial H}{\partial \boldsymbol{\lambda}}^\top = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{c}, \tag{22}$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{y}}^\top = -\boldsymbol{A}^\top\boldsymbol{\lambda}. \tag{23}$$

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

$$\left.\frac{\partial H}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}=\boldsymbol{u}^*(t)}^\top = \boldsymbol{R}\,\boldsymbol{u}^*(t) + \boldsymbol{B}^\top\boldsymbol{\lambda} = \boldsymbol{0}, \tag{24}$$

and thus,

$$\boldsymbol{u}^*(t) = -\boldsymbol{R}^{-1}\boldsymbol{B}^\top\boldsymbol{\lambda}(t). \tag{25}$$

Since Eq. (23) is decoupled from Eq. (22), its solution can be found independently. It is

$$\boldsymbol{\lambda}(t) = \mathrm{e}^{\boldsymbol{A}^\top(t_f - t)}\boldsymbol{\lambda}(t_f), \tag{26}$$

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

To find $\boldsymbol{\lambda}(t_f)$, let us consider the closed-form solution of Eq. (22) for $\boldsymbol{u} = \boldsymbol{u}^*(t)$:

$$\boldsymbol{y}(t) = \mathrm{e}^{\boldsymbol{A}t}\boldsymbol{y}(0) + \int_0^t \mathrm{e}^{\boldsymbol{A}(t-\tau)}\,(\boldsymbol{B}\boldsymbol{u}^*(\tau) + \boldsymbol{c})\,\mathrm{d}\tau. \tag{27}$$

If we evaluate this solution for $t = t_f$ and take into account Eqs. (25) and (26), we arrive at the expression

$$\boldsymbol{y}(t_f) = \boldsymbol{r}(t_f) - \boldsymbol{G}(t_f)\,\boldsymbol{\lambda}(t_f), \tag{28}$$

where

$$\boldsymbol{r}(t_f) = \mathrm{e}^{\boldsymbol{A}t_f}\,\boldsymbol{y}(0) + \int_0^{t_f} \mathrm{e}^{\boldsymbol{A}(t_f-\tau)}\,\boldsymbol{c}\,\mathrm{d}\tau, \tag{29}$$

and

$$\begin{aligned}\boldsymbol{G}(t_f) &= \int_0^{t_f} \mathrm{e}^{\boldsymbol{A}(t_f-\tau)}\,\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top\,\mathrm{e}^{\boldsymbol{A}^\top(t_f-\tau)}\,\mathrm{d}\tau \\ &= \int_0^{t_f} \mathrm{e}^{\boldsymbol{A}\tau}\,\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top\,\mathrm{e}^{\boldsymbol{A}^\top\tau}\,\mathrm{d}\tau.\end{aligned} \tag{30}$$

Given that $\boldsymbol{y}(t_f)$ is known from Eq. (20), we can solve Eq. (28) for $\boldsymbol{\lambda}(t_f)$ to obtain

$$\boldsymbol{\lambda}(t_f) = \boldsymbol{G}(t_f)^{-1}\,(\boldsymbol{r}(t_f) - \boldsymbol{y}(t_f))\,. \tag{31}$$

Now, substituting Eq. (31) into (26), and the result into Eq. (25), we finally obtain the optimal control policy for the fixed final state and fixed final time problem:

$$\boldsymbol{u}^*(t) = -\boldsymbol{R}^{-1}\boldsymbol{B}^\top\mathrm{e}^{\boldsymbol{A}^\top(t_f-t)}\,\boldsymbol{G}(t_f)^{-1}\,(\boldsymbol{r}(t_f) - \boldsymbol{y}(t_f))\,. \tag{32}$$

Note that this is an open-loop policy, as $\boldsymbol{u}^*$ depends on $t$ only. The values $\boldsymbol{r}(t_f)$ and $\boldsymbol{G}(t_f)$ in this policy can be obtained by computing the integrals in Eqs. (29) and (30) numerically. The matrix $\boldsymbol{G}(t_f)$ is known as the weighted continuous reachability Gramian, and since the system is controllable, it is symmetric and positive-definite for $t > 0$ [69], which ensures that $\boldsymbol{G}(t_f)^{-1}$ always exists.

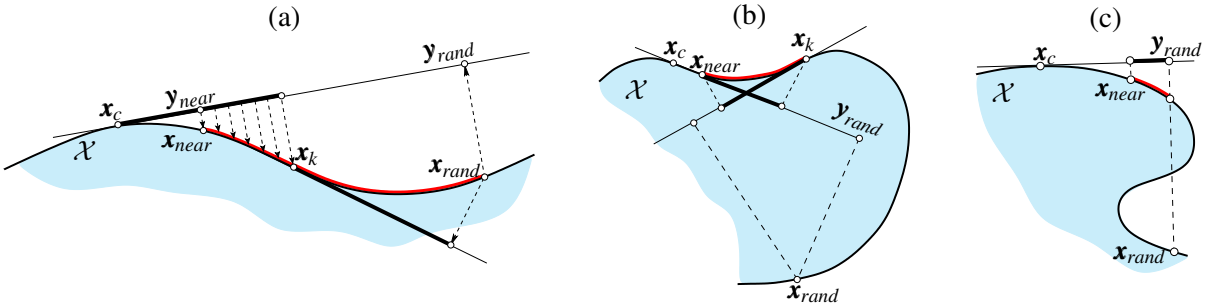



Figure 8: (a) Steering towards states not covered by the chart of $\boldsymbol{x}_{near}$. (b) Cyclic behavior of the steering method. (c) Convergence to $\boldsymbol{y}_{rand}$ but not to $\boldsymbol{x}_{rand}$.

### 6.4 Finding the optimal time $t_f$

To find a time $t_f^*$ for which the cost $J$ in Eq. (17) attains a minimum value, we substitute the optimal policy in Eq. (32) into Eq. (17), and take into account Eq. (30), obtaining

$$J(t_f) = t_f + [\boldsymbol{y}(t_f) - \boldsymbol{r}(t_f)]^\top \; \boldsymbol{G}(t_f)^{-1} \; [\boldsymbol{y}(t_f) - \boldsymbol{r}(t_f)] \,. \tag{33}$$

The time $t_f^*$ is thus the one that minimizes $J(t_f)$ in Eq. (33). Assuming that $t_f^*$ lies inside a specified time window $[0, t_{max}]$, this time can be computed approximately by evaluating $\boldsymbol{r}(t_f)$, $\boldsymbol{G}(t_f)$ and $J(t_f)$ using Eqs. (29), (30), and (33) 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 $\boldsymbol{G}(t_f^*)$ can be used to evaluate the optimal control policy in Eq. (32). By applying this policy to the full nonlinear system of Eq. (6) during $t_f^*$ seconds, we will follow a trajectory ending in some state $\boldsymbol{y}'_{rand}$ close to $\boldsymbol{y}_{rand}$. This trajectory can be recovered on the $\mathcal{X}$ space by means of the $\boldsymbol{\psi}_c$ map and, if it lies in $\mathcal{X}_{\text{feas}}$, the corresponding branch can be added to the RRT.

### 6.5 Steering over multiple charts

If $\boldsymbol{x}_{rand}$ is not covered by the chart $c$ of $\boldsymbol{x}_{near}$, we can iteratively apply the steering process as shown in Fig. 8(a). To this end, we compute $\boldsymbol{y}_{rand} = \boldsymbol{\varphi}_c(\boldsymbol{x}_{rand})$ and drive the system from $\boldsymbol{y}_{near} = \boldsymbol{\varphi}_c(\boldsymbol{x}_{near})$ towards $\boldsymbol{y}_{rand}$ on $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$, projecting the intermediate states $\boldsymbol{y}$ to $\mathcal{X}$ via $\boldsymbol{\psi}_c$. 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 Sec. 5.2). At this point, we generate a chart at $\boldsymbol{x}_k$ and linearise the system again. We then use this linearisation to recompute the optimal control policy to go from $\boldsymbol{x}_k$ to $\boldsymbol{x}_{rand}$. Such a "linearise and steer" process can be repeated as needed, until the system gets closely enough to $\boldsymbol{x}_{rand}$.

Although the previous procedure is in general effective, it can also fail in some situations. As shown in Fig. 8(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 $\boldsymbol{x}_{rand}$, a new control policy computed at $\boldsymbol{x}_k$ would steer the system back to $\boldsymbol{x}_{near}$, leading to a back-and-forth cycle not converging to $\boldsymbol{x}_{rand}$. Such limit cycles can be detected however, because the time $t_f^*$ will no longer decrease eventually. As shown in Fig. 8(c), moreover, the steering procedure can sometimes reach $\boldsymbol{y}_{rand}$, but we might find that $\boldsymbol{\psi}_c(\boldsymbol{y}_{rand}) \neq \boldsymbol{x}_{rand}$ because, due to the curvature of $\mathcal{X}$, several states can project to the same point on a given tangent space. Such situations do not prevent the connection of $\boldsymbol{x}_{near}$ with $\boldsymbol{x}_{rand}$ though, as the steering algorithm is to be used inside a higher-level RRT planner. The implementation of such a planner is addressed in the following section.

## 7 Planner implementation

Algorithm 1 gives the top-level pseudocode of the planner. At this level, the algorithm is almost identical to the RRT planner in [42]. 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. The atlas is initialized with one chart centred at $\boldsymbol{x}_s$ and another chart centred at $\boldsymbol{x}_g$ (line 1). As in [42], 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). Initially, a random state is sampled ($\boldsymbol{x}_{rand}$ 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 $\boldsymbol{x}_{rand}$ (line 7). The CONNECT method reaches a state $\boldsymbol{x}_{new}$ and adds it to $T_s$ if $\boldsymbol{x}_{new} \in \mathcal{X}_{\text{feas}}$. Due to the presence of obstacles or to a failure of the steering procedure, $\boldsymbol{x}_{new}$ may be different from $\boldsymbol{x}_{rand}$. Next, 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). This extension generates a new state $\boldsymbol{x}'_{new}$ that is added to $T_g$. 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), steps 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). Different metrics can be used to determine the distance between two states without affecting the overall structure of the planner. As in [42], we use the Euclidean distance for simplicity.

## 7.1 Sampling

The SAMPLE method is described in Algorithm 2. Initially, one of the charts covering the tree $T$ is selected at random with uniform distribution (line 2). A vector $\boldsymbol{y}_{rand}$ of parameters is next randomly sampled inside a ball of radius $\sigma$ centered at the origin of $\mathbb{R}^{d_\mathcal{X}}$ (line 3), repeating this sampling if necessary until $\boldsymbol{y}_{rand}$ falls inside the $P_c$ set for the selected chart. The method then attempts to compute the point $\boldsymbol{x}_{rand} = \boldsymbol{\psi}_c(\boldsymbol{y}_{rand})$ (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 $\boldsymbol{y}_{rand}$ (line 7). This point lies on $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$, instead of on $\mathcal{X}$, but it also provides a guiding direction to steer the tree towards uncharted regions of $\mathcal{X}$.

As explained in Sec. 5.3, the $P_c$ set of fully-surrounded charts become significantly smaller than the original ball of radius $\rho$, which decreases considerably their probability of being sampled. Charts that lie at the borders of the atlas, on the contrary, have fewer neighbouring charts (and thus a larger $P_c$ set), resulting in a higher probability of being sampled. In this way, the growing of the tree is biased towards uncharted regions of the state space.

**Algorithm 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 $\quad \boldsymbol{x}_{rand} \leftarrow$ SAMPLE($A, T_s$)
6 $\quad \boldsymbol{x}_{near} \leftarrow$ NEARESTSTATE($T_s, \boldsymbol{x}_{rand}$)
7 $\quad \boldsymbol{x}_{new} \leftarrow$ CONNECT($A, T_s, \boldsymbol{x}_{near}, \boldsymbol{x}_{rand}$)
8 $\quad \boldsymbol{x}'_{near} \leftarrow$ NEARESTSTATE($T_g, \boldsymbol{x}_{new}$)
9 $\quad \boldsymbol{x}'_{new} \leftarrow$ CONNECT($A, T_g, \boldsymbol{x}'_{near}, \boldsymbol{x}_{new}$)
10 $\quad$ 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 2:** Generate a random state $\boldsymbol{x}_{rand}$.

SAMPLE$(A,T)$
**input :** The atlas $A$ and the tree $T$ to be extended.
**output:** A guiding sample $\boldsymbol{x}_{rand}$.

1 **repeat**
2 $\quad c \leftarrow$ RANDOMCHARTINDEX$(A,T)$
3 $\quad \boldsymbol{y}_{rand} \leftarrow$ RANDOMONBALL$(\sigma)$
4 **until** $\boldsymbol{y}_{rand} \in P_c$
5 $\boldsymbol{x}_{rand} \leftarrow \boldsymbol{\psi}_c(\boldsymbol{y}_{rand})$
6 **if** $\boldsymbol{x}_{rand} =$ NULL **then**
7 $\quad \boldsymbol{x}_{rand} \leftarrow \boldsymbol{x}_c + \boldsymbol{U}_c \boldsymbol{y}_{rand}$
8 RETURN$(\boldsymbol{x}_{rand})$

**Algorithm 3:** Try to connect $\boldsymbol{x}_{near}$ with $\boldsymbol{x}_{rand}$.

CONNECT$(A,T,\boldsymbol{x}_{near},\boldsymbol{x}_{rand})$
**input :** An atlas $A$, a tree $T$, the state $\boldsymbol{x}_{near}$ from which $T$ is to be extended, and the guiding sample $\boldsymbol{x}_{rand}$.
**output:** The new state $\boldsymbol{x}_{new}$.

1 $\boldsymbol{x}_{new} \leftarrow \boldsymbol{x}_{near}$
2 $t^*_{fp} \leftarrow \infty$
3 **repeat**
4 $\quad (\boldsymbol{u}^*, t^*_f) \leftarrow$ LQRPOLICY$(A,\boldsymbol{x}_{near},\boldsymbol{x}_{rand})$
5 $\quad$ **if** $t^*_f \leq t^*_{fp}$ **then**
6 $\quad\quad t^*_{fp} \leftarrow t^*_f$
7 $\quad\quad (\boldsymbol{x}_{new},\boldsymbol{u}_{new}) \leftarrow$ SIMULATE$(A,T,\boldsymbol{x}_{near},\boldsymbol{x}_{rand},\boldsymbol{u}^*,t^*_f)$
8 $\quad\quad$ **if** $\boldsymbol{x}_{new} \in \mathcal{X}_{feas}$ **then**
9 $\quad\quad\quad T \leftarrow$ ADDEDGE$(T,\boldsymbol{x}_{near},\boldsymbol{u}_{new},\boldsymbol{x}_{new})$
10 $\quad\quad\quad \boldsymbol{x}_{near} \leftarrow \boldsymbol{x}_{new}$
11 **until** $\boldsymbol{x}_{new} \notin \mathcal{X}_{feas}$ **or** $\|\boldsymbol{x}_{new} - \boldsymbol{x}_{rand}\| \leq \delta$ **or** $t^*_f > t^*_{fp}$
12 RETURN$(\boldsymbol{x}_{new})$

## 7.2 Tree extension

Algorithm 3 attempts to connect a state $\boldsymbol{x}_{near}$ to a state $\boldsymbol{x}_{rand}$. The algorithm implements a loop where, initially, the optimal policy $\boldsymbol{u}^*$ and time $t^*_f$ to connect these two states are computed (line 4). The policy is a function of time given by Eq. (32). If $t^*_f$ is lower than the optimal time $t^*_{fp}$ obtained in the previous iteration, the policy is used to simulate the evolution of the system from $\boldsymbol{x}_{near}$ (line 7). The simulation produces a new state $\boldsymbol{x}_{new}$ which, if feasible (i.e., if it is collision-free and inside the workspace limits), is added to the tree. This involves the creation of an edge between $\boldsymbol{x}_{near}$ and $\boldsymbol{x}_{new}$ (line 9), which also stores the control sequence $\boldsymbol{u}_{new}$ executed in the simulation. The loop is repeated until the new state is unfeasible, or $\boldsymbol{x}_{rand}$ is reached with accuracy $\delta$, or $t^*_f$ is larger than $t^*_{fp}$ (line 11).

Algorithm 4 summarizes the procedure used to simulate a given policy $\boldsymbol{u}^*(t)$ from a particular state $\boldsymbol{x}_k$. The simulation is carried on while the new state is valid, and the target state is not reached with accuracy $\delta$, and the integration time $t$ is lower than $t^*_f$ (line 4). A state is not valid if is not in $\mathcal{X}_{\text{feas}}$ (line 9), if it is not in the validity area of the chart (line 15), or if it is not included in the current $P_c$ set (line 21). In the first case, both the simulation and the connection between states are stopped. In the last two cases the simulation is stopped, but the connection continues after recomputing the optimal policy, either using a newly-created chart (line 14) or the atlas chart covering the new state.

The key procedure in the simulation is the NEXTSTATE method (line 7), which provides the next

**Algorithm 4:** Simulate an action.

SIMULATE($A, T, \boldsymbol{x}_k, \boldsymbol{x}_{rand}, \boldsymbol{u}^*, t_f^*$)
**input :** An atlas, $A$, a tree, $T$, the state from where to start the simulation, $\boldsymbol{x}_k$, the state to approach, $\boldsymbol{x}_{rand}$, the policy to simulate, $\boldsymbol{u}^*$, and the optimal time $t_f^*$ to simulate.
**output:** The last state in the simulation and the executed control sequence.

$t \leftarrow 0$
$\boldsymbol{u}_k \leftarrow \emptyset$
VALIDSTATE $\leftarrow$ TRUE
**while** VALIDSTATE **and** $\|\boldsymbol{x}_k - \boldsymbol{x}_{rand}\| > \delta$ **and** $|t| < t_f^*$ **do**
  $c \leftarrow$ CHARTINDEX($\boldsymbol{x}_k$)
  $\boldsymbol{y}_k \leftarrow \boldsymbol{\varphi}_c(\boldsymbol{x}_k)$
  $(\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}, h) \leftarrow$ NEXTSTATE($\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{u}^*(t), \boldsymbol{F}, \boldsymbol{x}_c, \boldsymbol{U}_c, \delta$)
  **if** $\boldsymbol{x}_{k+1} \notin \mathcal{X}_{feas}$ **then**
    VALIDSTATE $\leftarrow$ FALSE
  **else**
    **if** $\|\boldsymbol{x}_{k+1} - (\boldsymbol{x}_c + \boldsymbol{U}_c \boldsymbol{y}_{k+1})\| > \varepsilon$ **or**
      $\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\| / \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| < \cos(\alpha)$ **or**
      $\|\boldsymbol{y}_{k+1}\| > \rho$ **then**
      ADDCHARTTOATLAS($A, \boldsymbol{x}_k$)
      VALIDSTATE $\leftarrow$ FALSE
    **else**
      $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k+1}$
      $\boldsymbol{u}_k \leftarrow \boldsymbol{u}_k \cup \{(\boldsymbol{u}(t), h)\}$
      $t \leftarrow t + h$
      **if** $\boldsymbol{y}_{k+1} \notin P_c$ **then**
        VALIDSTATE $\leftarrow$ FALSE
RETURN($\boldsymbol{x}_k, \boldsymbol{u}_k$)

state $\boldsymbol{x}_{k+1}$, given the current state $\boldsymbol{x}_k$ and the action $\boldsymbol{u}^*(t)$ at time $t$. This is implemented by integrating Eq. (6) using local coordinates as explained in Section 5.1. Any numerical integration method, either explicit or implicit, could be used to discretise Eq. (10). 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 [57]. Also, it gives more stable and accurate solutions over long time intervals. Using this rule, Eq. (10) is discretised as

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{h}{2} \boldsymbol{U}_c^\top \left( \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}) + \boldsymbol{g}(\boldsymbol{x}_{k+1}, \boldsymbol{u}) \right), \tag{34}$$

where $h$ is the integration time step. The value $\boldsymbol{x}_{k+1}$ in Eq. (34) is unknown but, since it must satisfy Eq. (9), it must fulfil

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{x}_{k+1}) &= \boldsymbol{0}, \\ \boldsymbol{U}_c^\top (\boldsymbol{x}_{k+1} - \boldsymbol{x}_c) - \boldsymbol{y}_{k+1} &= \boldsymbol{0}. \end{aligned} \tag{35}$$

Now, substituting Eqs. (34) into Eq. (35) we obtain

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{x}_{k+1}) &= \boldsymbol{0}, \\ \boldsymbol{U}_c^\top (\boldsymbol{x}_{k+1} - \tfrac{h}{2}(\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}) + \boldsymbol{g}(\boldsymbol{x}_{k+1}, \boldsymbol{u})) - \boldsymbol{x}_c) - \boldsymbol{y}_k &= \boldsymbol{0}, \end{aligned} \tag{36}$$

where $\boldsymbol{x}_k$, $\boldsymbol{y}_k$, and $\boldsymbol{x}_c$ are known 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 more suitable as it avoids the computation of the Jacobian of the system at each step. Potra and Yen [57] gave an approximation of this Jacobian that allows finding $\boldsymbol{x}_{k+1}$ in only a few iterations.

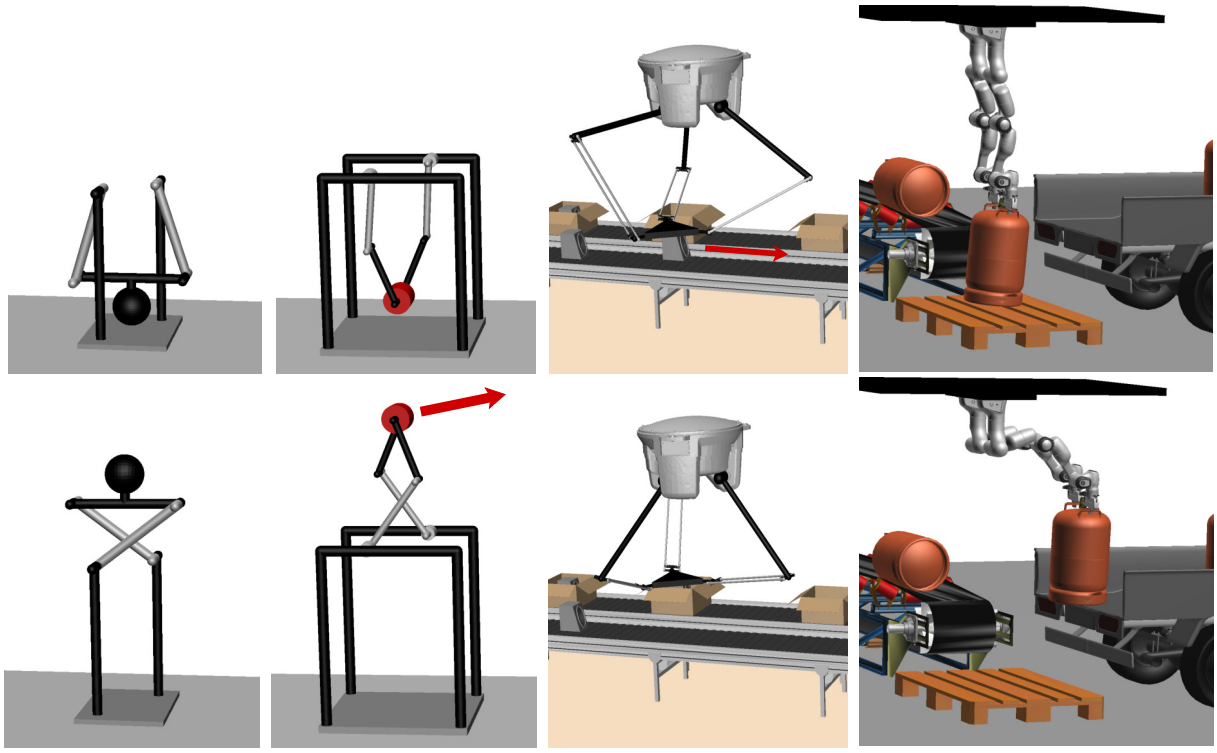


Figure 9: Example tasks used to illustrate the performance of the planner. From left to right, and columnwise: weight lifting, weight throwing, conveyor switching, and truck loading. The robots involved are, respectively, a four-bar robot, a five-bar robot, a Delta robot, and a double-arm manipulation system. The top and bottom rows show the start and goal states for each task. In the goal state of the second task, and in the start state of the third task, the load is moving at a certain velocity indicated by the red arrow. The velocity of the remaining start and goal states is null. In all robots, the motor torques are limited to prevent the generation of direct trajectories to the goal.

Table 1: Problem dimensions and performance statistics for the example tasks.

| | | | | | Randomized steering | | | | LQR steering | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Example task** | $n_q$ | $n_e$ | $d_\mathcal{X}$ | $n_u$ | **No. samples** | **No. charts** | **Plan. Time (s)** | **Success Rate** | **No. samples** | **No. charts** | **Plan. Time (s)** |
| Weight lifting | 4 | 3 | 2 | 1 | 875 | 160 | 0.7 | 100% | 128 | 95 | 0.4 |
| Weight throwing | 5 | 3 | 4 | 2 | 12837 | 2295 | 229.6 | 100% | 717 | 242 | 8.5 |
| Conveyor switching | 15 | 12 | 6 | 3 | 38597 | 313 | 561.4 | 100% | 4889 | 173 | 88.8 |
| Truck loading | 10 | 6 | 8 | 10 | 9381 | 1087 | 1930.0 | 45% | 6588 | 1103 | 104.0 |

For backward integration, i.e., when extending the RRT with root at $\boldsymbol{x}_g$, the time step $h$ in Eq. (36) mas be negative negative. In any case, $h$ is adjusted so that the change in parameter space, $\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\|$, is bounded by $\delta$, with $\delta \ll \rho$. This is necessary to accurately detect the transitions between charts.

## 8 Planning examples

The planner has been implemented in C and it has been integrated into the CUIK suite [55]. We next analyse its performance in planning four tasks of increasing complexity (Fig. 9). The first two tasks involve planar single-loop mechanisms, which are simple enough to illustrate key aspects of the planner, like the formulation of Eqs. (1) and (2), the performance of the steering method, the traversal of singularities, or the ability to plan trajectories towards nonzero velocity states. The third and fourth tasks, on the other hand, show the planner performance in spatial robots of considerable complexity. In all cases

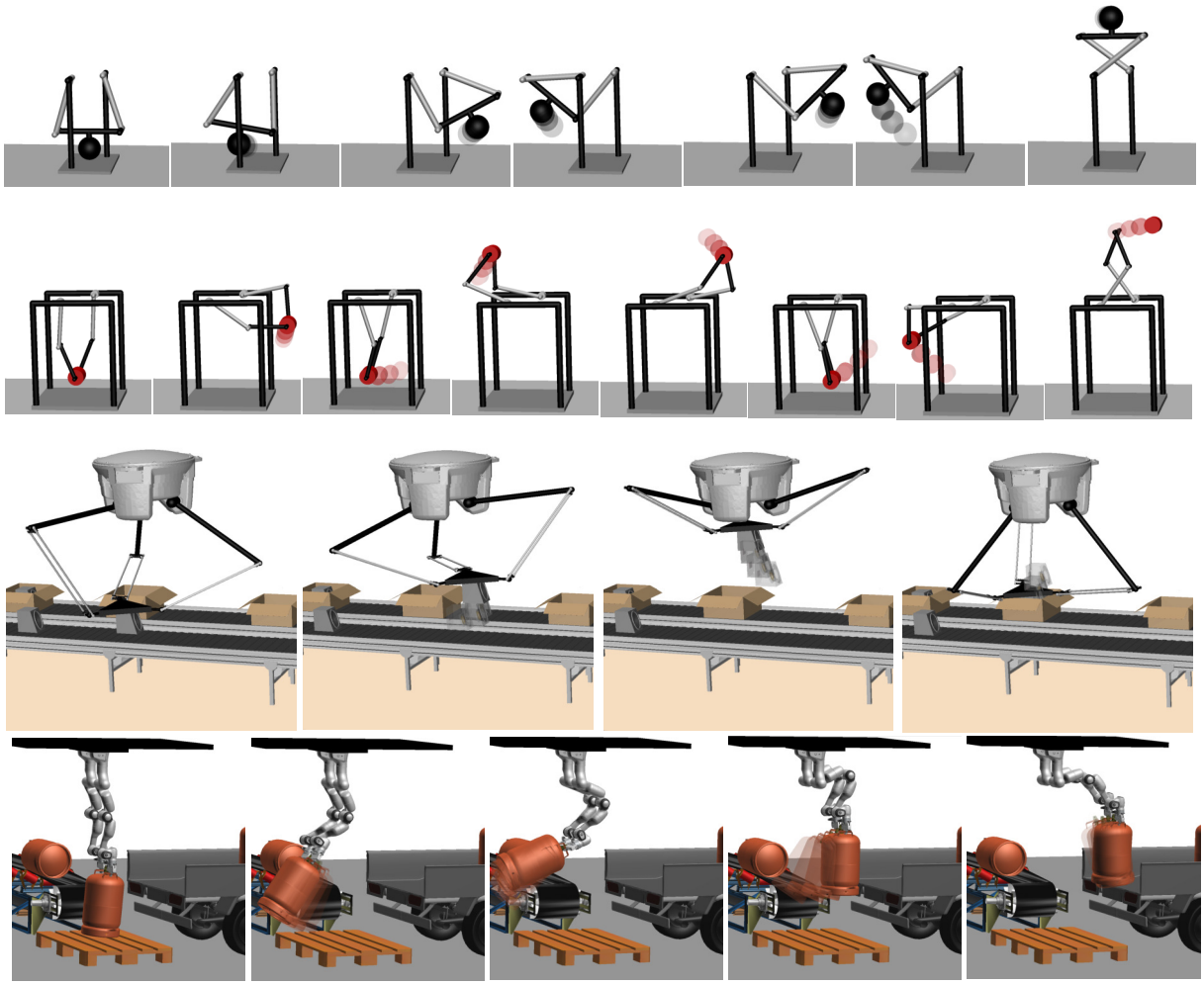


Figure 10: Solution trajectories for the four test cases. The shown trails depict earlier positions of the load during a same time span. A longer trail, therefore, corresponds to a higher velocity of the load. See youtu.be/-_DMzK5SGzQ for an animated version of this figure.

the robots are subject to gravity and viscous friction in all joints, and their action bounds $l_i$ in Eq. (7) are small enough so as to impede direct trajectories between $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$. This complicates the problems and forces the generation of swinging motions to reach the goal. As for the planner parameters, the matrix $\boldsymbol{R}$ in Eq. (17) has been set to be diagonal with $\boldsymbol{R}_{i,i} = 1/l_i^2$, and we have fixed $\delta = 0.05$, $\sigma = d_{\mathcal{X}}$, $\rho = d_{\mathcal{X}}/2$, $\cos(\alpha) = 0.1$, $\varepsilon = 0.1$, $\beta = 0.05 d_{\mathcal{X}}$, and $t_m = 1$. No special effort has been made to tune these parameters however, since the performance gracefully degrades when modifying their values. The resulting trajectories can be seen in Fig. 10 and in the companion video of this report (also available through youtu.be/-_DMzK5SGzQ. The complete set of geometric and dynamic parameters of all examples are provided in http://www.iri.upc.edu/cuik.

Table 1 summarizes the problem dimensions and performance statistics for the four mentioned tasks. 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 ($n_u$). The table also provides the average over twenty runs of the number of samples and charts required to solve the problem, and the planning time in seconds using a MacBook Pro with an Intel i9 octa-core processor running at 2.93 GHz. This time is largely dependent on $d_{\mathcal{X}}$ and $n_u$, but is also affected by many other

aspects, like the torque limits of the actuators, the system masses, or the presence of obstacles. Note also that the number of charts needed to plan a task does not necessarily grow in parallel with the planning time, as it mostly depends on the curvature and size of the region of $\mathcal{X}$ that has to be explored in order to solve the problem. In the table, statistics for both the randomized steering strategy in [14] and the LQR steering strategy we propose are given for comparison. The randomized strategy employs $2\,n_u$ random actions from $\mathcal{U}$, and its parameter value $t_m$ is set to 0.1 in accordance with [14]. As seen in the table, in terms of exploration the randomized strategy is less efficient than the LQR strategy, as it requires a larger number of samples and charts to find a solution. Although the randomized strategy has been implemented by simulating the various random actions in parallel, it is still slower than the LQR strategy. In fact, the success rate of the randomised strategy is only 45% in the truck loading task (fixing a maximum planning time of one hour). Further details on the four tasks are next provided.

## 8.1 Weight lifting

The first task to be planned consists in lifting a heavy load with a four-bar robot (Fig. 9, left column). The robot involves four links cyclically connected with revolute joints (Fig. 11). Following Appendix B, we label the links as $L_0,\ldots,L_3$, 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 then given by $\boldsymbol{q} = (q_1, q_2, q_3, q_4)$.

To formulate Eq. (1), we attach a coordinate system to each link $L_i$, centred at joint $J_{i+1}$ and with the $x_i$ axis aligned with the link. This system is called the link $i$ coordinates. The loop-closure condition of robot can then be written as

$$\prod_{i=1}^{4} \boldsymbol{T}_z(q_i) \cdot \boldsymbol{C}_i = \boldsymbol{I}, \tag{37}$$

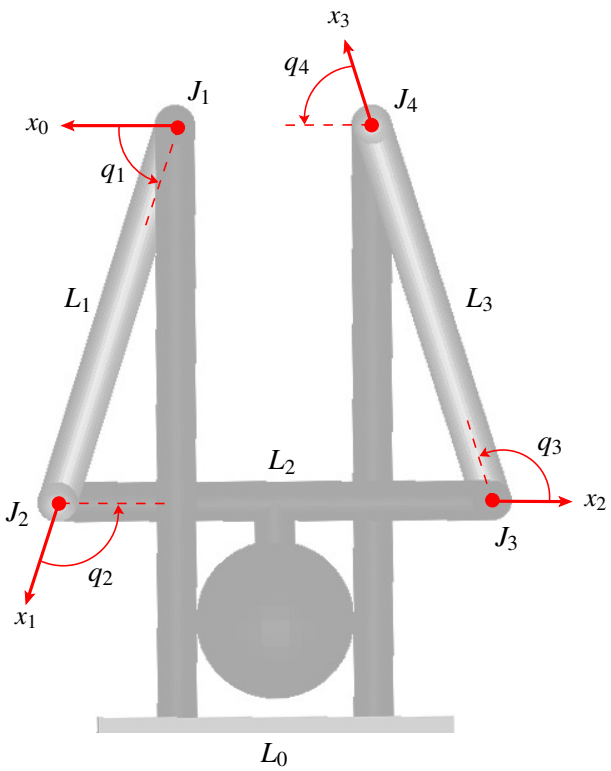



Figure 11: Geometry of the four-bar mechanism in Fig. 9, left column. For each coordinate system, only the $x$ axis is depicted.

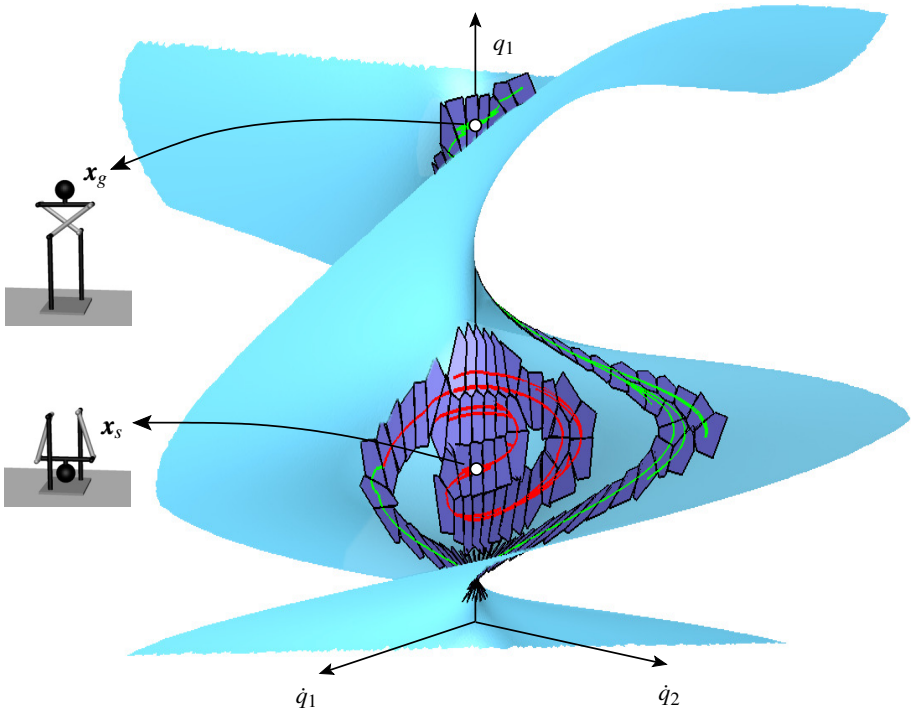



Figure 12: A partial atlas of $\mathcal{X}$ used to plan the lifting of a weight with the four-bar robot. The red and green trees are rooted at $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$ respectively, and they are grown towards each other in parallel with the atlas. Each polygon in dark blue corresponds to the $P_c$ set of a given chart. See youtu.be/-_DMzK5SGzQ for an animated version of this figure.

where

$$\boldsymbol{T}_z(q_i) = \begin{bmatrix} \cos(q_i) & -\sin(q_i) & 0 \\ \sin(q_i) & \cos(q_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{38}$$

and

$$\boldsymbol{C}_i = \begin{bmatrix} 1 & 0 & d_i \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{39}$$

are the planar counterparts of the transforms in Eq. (46), $d_i$ is the distance between the two revolute joints of link $L_i$, and $\boldsymbol{I}$ is the $3 \times 3$ identity matrix. To form Eq. (1), however, it suffices to select the scalar equations that correspond to the elements $(2,1)$, $(1,3)$, and $(2,3)$ of Eq. (37), as the remaining equations are dependent on them.

Eq. (2) could now be obtained by taking the time derivative of Eq. (1), but Appendix B shows that this equation reduces to $\boldsymbol{J} \cdot \dot{\boldsymbol{q}} = 0$, where $\boldsymbol{J}$ is the screw Jacobian of the 4-bar loop. This Jacobian has the form

$$\boldsymbol{J} = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ -a_1 & -a_2 & -a_3 & -a_4 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \tag{40}$$

where $(a_i, b_i)$ are the $(x,y)$ coordinates of joint $J_i$ in link 0 coordinates [22]. Using the fact that $(a_1, b_1) = (0,0)$, these coordinates can be written as follows in terms of $q_1, \ldots, q_4$:

$$a_i = a_i - 1 + d_i \cos(q_1 + \ldots + q_{i-1}), \tag{41}$$

$$b_i = b_i - 1 + d_i \sin(q_1 + \ldots + q_{i-1}). \tag{42}$$

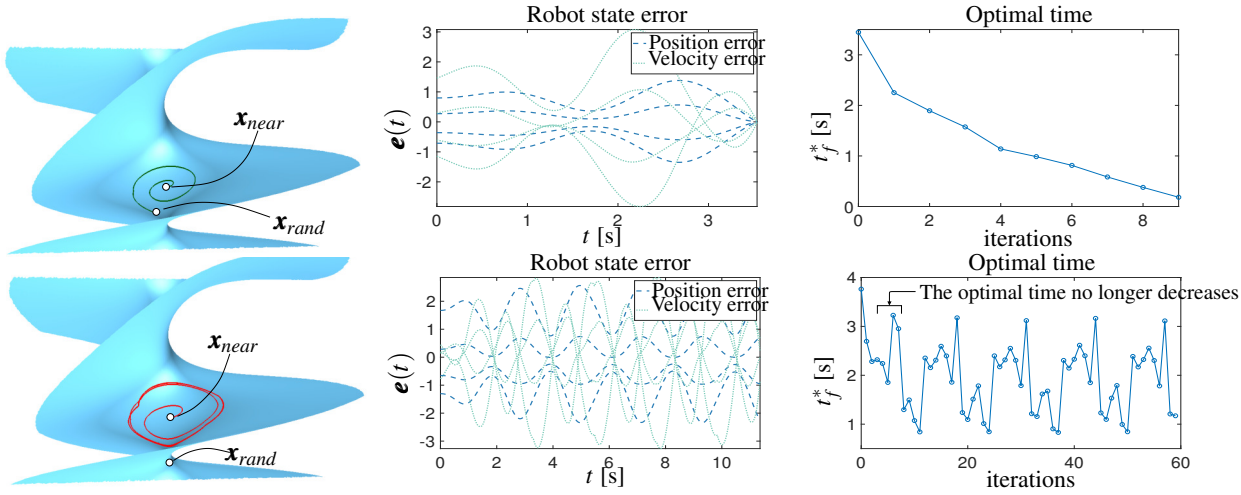



Figure 13: Steering the four-bar robot from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{rand}$. Top: The LQR strategy allows the planner to connect $\boldsymbol{x}_{near}$ and $\boldsymbol{x}_{rand}$. Bottom: The strategy enters a limit cycle and is never able to reach $\boldsymbol{x}_{rand}$. The right plot shows that $t_f^*$ no longer decreases after six iterations, so it would be aborted at this point.

Under the previous formulation we have $n_q = 4$ and $n_e = 3$, so in this case $\mathcal{X}$ is of dimension

$$d_{\mathcal{X}} = 2\, d_{\mathcal{C}} = 2\,(n_q - n_e) = 2.$$

To have an idea, Fig. 12 shows the shape of $\mathcal{X}$ when projected to the space defined by $q_1$, $\dot{q}_1$, and $\dot{q}_2$, 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 be able 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. 10, top row.

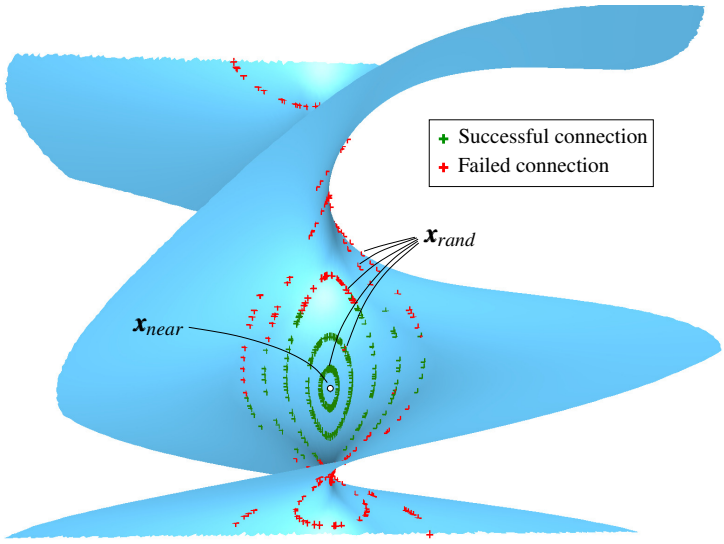
The same example can be used to illustrate the performance of the LQR steering strategy. Fig. 13-top, shows an example in which this strategy successfully finds a trajectory connecting $\boldsymbol{x}_{near}$ with $\boldsymbol{x}_{rand}$, with $t_f^*$ always decreasing. In contrast, Fig. 13-bottom shows another example in which the process tends to a limit cycle like the one in Fig. 8(b), and is never able to reach the goal. The steering method in Algorithm 3 would stop after a few iterations because a point is reached in which $t_f^*$ no longer decreases.

In Fig. 14 we also show the performance of the LQR strategy for states $\boldsymbol{x}_{rand}$ that are progressively further away from $\boldsymbol{x}_{near}$. 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 $\boldsymbol{x}_{near}$. As a reference, the distance from $\boldsymbol{x}_s$ to $\boldsymbol{x}_g$ is 3.7 in this example. The states $\boldsymbol{x}_{rand}$ that could be connected to $\boldsymbol{x}_{near}$ are shown in green, while those that could not are shown in red. As expected for a local planner, the closer $\boldsymbol{x}_{rand}$ from $\boldsymbol{x}_{near}$, the higher the probability of success of the steering process.

## 8.2 Weight throwing

The second task involves a five-bar robot. It consists in throwing a given object from a certain position at a prescribed velocity (indicated with the red arrow in Fig. 9, second column). This shows the planner ability to reach goal states $\boldsymbol{x}_g$ with nonzero velocity, which would be difficult to achieve with conventional C-space approaches.

The formulation of Eqs. (1) and (2) 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. Only the two ground joints are actuated, so $n_u = 2$.




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

Figure 14: Success rate of the LQR steering strategy for states $x_{rand}$ that are increasingly far from $x_{near}$.

The computed trajectory can be seen in the second row of Fig. 10. The robot first lifts the object to the right until it achieves a zero-velocity position (second snapshot), to later move it back to the left along a nearly-circular path (remaining snapshots). Almost two turns of this path are completed in order to reach the launch point with the required momentum (last snapshot).

The task also illustrates the planner capacity to traverse forward singularities, which are configurations in which the robot is locally underactuated. These configurations are difficult to manage, as they can only be crossed under very specific velocities and accelerations fulfilling certain rank-deficiency conditions [15, 16]. However, since our planner trajectories result from simulating control policies $\boldsymbol{u}(t)$ using forward dynamics, they naturally satisfy the mentioned conditions at the singularities, and are thus kinematically and dynamically feasible even in such configurations. In particular, a five-bar robot is known to exhibit a forward singularity when its two distal links happen to be aligned [15]. In the trajectory shown in Fig. 10 this occurs in the third and sixth snapshots. From the companion video we see that the robot passes through these configurations in a smooth and predictable manner with no difficulty. Note that, while such a trajectory would be difficult to execute using classical computed-torque controllers [1], recent LQR controllers for closed kinematic chains have no trouble in accomplishing this task [13].

## 8.3 Conveyor switching

In the previous tasks the robot was a single-loop mechanism in an obstacle-free environment. To exemplify the planner in a multi-loop mechanism surrounded by obstacles, we next apply it to a conveyor switching task on a Delta robot (Fig. 9, third column). 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. 15, left). 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 modelled with seven joint angles, we use the fact that the leg is kinematically equivalent to an *R*-*U*-*U* chain (Fig. 15, right), where *U* refers to a universal joint. By noting that 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

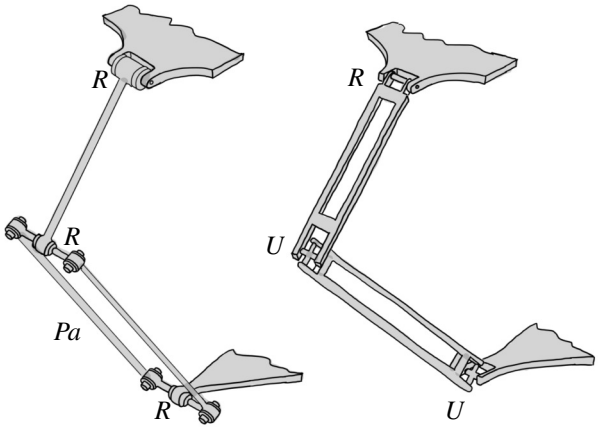



Figure 15: A leg of a Delta robot (left) and its equivalent *R-U-U* chain (right).

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 Eqs. (1) and (2), note that every two legs of the robot define a six-dimensional loop-closure constraint (Appendix C), which gives three such constraints in total. Only two of the constraints are actually independent however, 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, our 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. 10, third row, and in its companion video. Given the velocity of the moving belt, the planner is forced to reduce the initial momentum 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 on the goal location.

## 8.4 Truck loading

The fourth task involves two 7-DOF Franka Emika arms moving a gas bottle cooperatively. 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 policies 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.

The example also illustrates that the randomized steering strategy performs poorly when $n_u$ is large. 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, [42] proposes to simulate only $2\,n_u$ random actions for each branch extension. Our implementation adopts this criterion but, like [42], 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 1). We have also tried to simulate $2^{n_u}$ random actions, instead of just $2\,n_u$, 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 policy

per branch extension, so an increase in $n_u$ does not affect the planning time dramatically (Table 1, last column). Using this strategy, the planner obtained the trajectory shown in Fig. 10, bottom row, in which we see that, in order to gain momentum, the robot is moved backwards before it lifts the bottle onto the truck.

## 9 Conclusions

This report has proposed a randomised planner to compute dynamically-feasible trajectories for robots with closed kinematic chains. The state space of such robots is an intricate manifold that poses three major hurdles on the planner design: 1) the generation of random samples on the manifold; 2) the accurate simulation of robot trajectories along the manifold; and 3) the steering of the system towards random states. The three issues have been addressed by constructing an atlas of the manifold in parallel to the RRT. The result is a planner that can explore the state space in an effective manner, while conforming to the vector fields defined by the equations of motion of the robot and the force bounds of the actuators. The planner is probabilistically complete, but a proof of this point has been omitted because it would be lengthy, and it would mainly replicate the arguments in [39] with minor adaptations. The examples in the report show that the planner can solve significantly complex problems that require the computation of swinging motions between start and goal states, under restrictive torque limitations imposed on the motors.

Several points should be considered in further improvements of this work. Note that, as usual in a randomised planner, our control policies are piecewise continuous, so the planned trajectories are smooth in position, but not in velocity or acceleration. Therefore, to reduce control or vibration issues in practice, a post-processing should be applied to obtain twice-differentiable trajectories. The trajectories should also be made locally-optimal in some sense, minimising the time or control effort required for its execution. Trajectory optimization tools like those in [56], [4], or [5] might be very helpful to both ends. Another sensitive point is the metric employed to measure the distance between two states. This is a general concern in any motion planner, but it is more difficult to address in our context as such 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. Another point deserving attention would be the evaluation of constraint forces during the planning. While such forces result in no motion, they do stress the robot parts unnecessarily and should be kept below admissible bounds. The ability to impose bounds on the constraint forces would also allow the planner to compute trajectories in closed kinematic chains induced by unilateral contacts, like those that arise when a hand moves an object in contact with a surface.

# Appendices

## A The $\Psi$ mapping

Note that $\frac{\partial \boldsymbol{\psi}(\boldsymbol{y})}{\partial \boldsymbol{y}}$ can be computed in closed form. Consider the mapping

$$\boldsymbol{y} = \boldsymbol{\varphi}(\boldsymbol{x}) = \boldsymbol{U}_c^\top (\boldsymbol{x} - \boldsymbol{x}_c) . \tag{43}$$

Lets now use the inverse mapping $\boldsymbol{x} = \boldsymbol{\psi}(\boldsymbol{y})$ to rewrite Eq. (43) as

$$\boldsymbol{y} = \boldsymbol{U}_c^\top (\boldsymbol{\psi}(\boldsymbol{y}) - \boldsymbol{x}_c) . \tag{44}$$

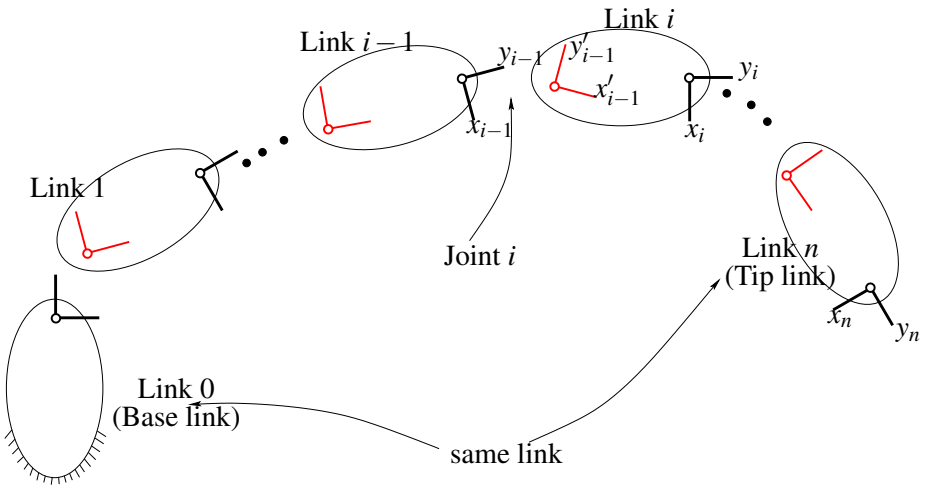



Figure 16: A kinematic loop in exploded view. The tip link is a copy of the base link. In an assembled configuration, these two bodies are forced to coincide.

If we compute the partial derivative of both side of Eq. (44) with respect to $\boldsymbol{y}$, we have

$$\boldsymbol{I}_{d_{\mathcal{X}}} = \boldsymbol{U}_c^{\top} \frac{\partial \boldsymbol{\psi}(\boldsymbol{y})}{\partial \boldsymbol{y}}, \tag{45}$$

where $\boldsymbol{I}_{d_{\mathcal{X}}}$ is the $d_{\mathcal{X}} \times d_{\mathcal{X}}$ identity matrix. Thus, $\frac{\partial \boldsymbol{\psi}(\boldsymbol{y})}{\partial \boldsymbol{y}}$ must be $\boldsymbol{U}_c$.

# B Formulation of the state space equations

The first step to formulate Eqs. (1) and (2) is to select appropriate $\boldsymbol{q}$ coordinates to define our system configuration. The main choices in multibody dynamics include relative coordinates, reference point coordinates, and natural coordinates [19]. In our case, we adopt relative coordinates, as they provide compact formulations that easily allow the modelling of actuation and friction forces at the joints. This appendix shows how to formulate Eqs. (1) and (2) using such coordinates. We assume for simplicity that the robot exhibits just one kinematic loop. If more loops were present, we would just collect the equations below for a maximal set of independent loops in the mechanism. We also treat revolute or prismatic joints only, as more complex joints can always be formulated using a combination of these elementary pairs.

A kinematic loop can be regarded as a serial chain in which the base and tip links are forced to coincide. Fig. 16 shows such a chain in exploded view, with our conventions depicted. The links are numbered from 0 to $n$, where 0 and $n$ refer to the base and tip links respectively. The joints are numbered from 1 to $n$, with joint $i$ being the one that connects links $i$ and $i-1$. We also define two coordinate systems for each joint: one attached to the body that is closest to the base, and one attached to the body that is closest to the tip. On joint $i$, these two systems are called the $(i-1)$ and $(i-1)'$ coordinates respectively. The tip link also has another coordinate system attached, called the link $n$ coordinates, which should coincide with the link 0 coordinates in an assembled configuration.

With the previous definitions, the homogeneous transformation that locates the link $i$ coordinates relative to the link $i-1$ ones is given by

$${}^{i-1}\boldsymbol{T}_i = \boldsymbol{T}_z(q_i) \cdot \boldsymbol{C}_i \tag{46}$$

where $\boldsymbol{T}_z(q_i)$ is given in Table 2, and $\boldsymbol{C}_i$ is a constant transformation that locates the $i$ coordinates relative

to the $(i-1)'$ ones. Then, by defining

$$\boldsymbol{T}(\boldsymbol{q}) = {}^{0}\boldsymbol{T}_1 \cdot {}^{1}\boldsymbol{T}_2 \cdot \ldots \cdot {}^{n-1}\boldsymbol{T}_n, \tag{47}$$

the loop-closure constraint can simply be posed as

$$\boldsymbol{T}(\boldsymbol{q}) = \boldsymbol{I}. \tag{48}$$

Eq. (48) is a system of $4 \times 4$ scalar equations in $n$ unknowns, but only six of these equations are independent [58]. In particular, Eq. (1) can be formulated by choosing

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{T}_{3,2} \\ \boldsymbol{T}_{1,3} \\ \boldsymbol{T}_{2,1)} \\ \boldsymbol{T}_{1,4} \\ \boldsymbol{T}_{2,4} \\ \boldsymbol{T}_{3,4} \end{bmatrix} \tag{49}$$

where $\boldsymbol{T}_{i,j}$ refers to the $(i,j)$ entry of $\boldsymbol{T}(\boldsymbol{q})$.

On the other hand, the feasible velocities $\dot{\boldsymbol{q}} = (\dot{q}_1, \ldots, \dot{q}_n)$ are those that fulfil the time derivative of Eq. (48):

$$\frac{\partial \boldsymbol{T}}{\partial q_1} \cdot \dot{q}_1 + \ldots + \frac{\partial \boldsymbol{T}}{\partial q_n} \cdot \dot{q}_n = \boldsymbol{0}. \tag{50}$$

From the results in [67], it is not difficult to see that, when $\boldsymbol{q}$ satisfies Eq. (48), we have

$$\frac{\partial \boldsymbol{T}}{\partial q_i} = \boldsymbol{S}_i,$$

where $\boldsymbol{S}_i$ is the $4 \times 4$ matrix encoding the unit twist of link $i$ as observed from link $i-1$, but expressed in link 0 coordinates. This twist can be obtained by using the change of coordinates formula

$$\boldsymbol{S}_i = {}^{0}\boldsymbol{T}_{i-1} \cdot {}^{i-1}\boldsymbol{S}_i \cdot {}^{0}\boldsymbol{T}_{i-1}^{-1},$$

where ${}^{i-1}\boldsymbol{S}_i$ is the same twist, but expressed in link $i-1$ coordinates (see Table 2). Like in Eq. (48), only six of the $4 \times 4$ equations of Eq. (50) are actually independent. From these equations, we can extract a subset of six independent equations by writing

$$\hat{\boldsymbol{S}}_1 \cdot \dot{q}_1 + \cdots + \hat{\boldsymbol{S}}_n \cdot \dot{q}_n = 0, \tag{51}$$

where $\hat{\boldsymbol{S}}_i \in \mathbb{R}^6$ is the spatial velocity vector that corresponds to the $\boldsymbol{S}_i$ matrix. This vector is the one formed by taking the components $(3,2)$, $(1,3)$, $(2,1)$, $(1,4)$, $(2,4)$, and $(3,4)$ of $\boldsymbol{S}_i$. Therefore, when formulating loop-closure equations, Eq. (2) can be written as

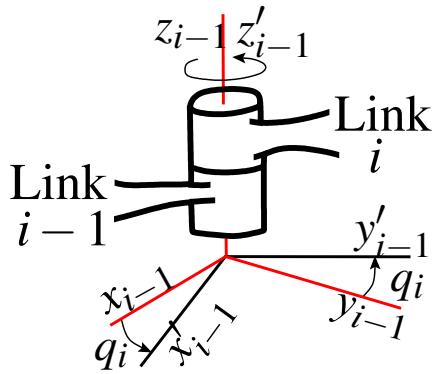
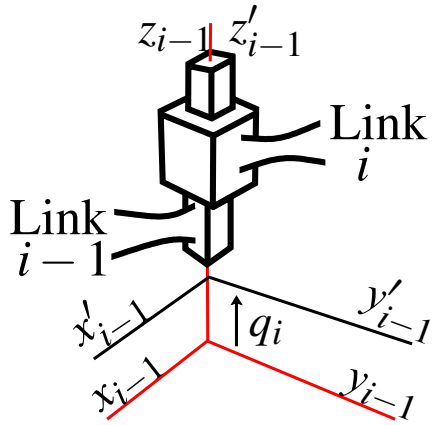
$$\boldsymbol{J} \cdot \dot{\boldsymbol{q}} = 0, \tag{52}$$

where $\boldsymbol{J} = [\hat{\boldsymbol{S}}_1, \ldots, \hat{\boldsymbol{S}}_n]$ is the screw Jacobian of the kinematic loop [24]. In other words, $\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q}) = \boldsymbol{J}$ when $\boldsymbol{\Phi}(\boldsymbol{q}) = \boldsymbol{I}$ and $\boldsymbol{\Phi}(\boldsymbol{q})$ is defined as in Eq. (49).

# C Formulation of the equation of motion

We now turn into the problem of formulating Eq. (6). As in Appendix B, we assume that our robot consists of a single kinematic loop with $n$ 1-DOF joints, which can be prismatic or revolute joints. For convenience, such a loop can be viewed as being cut at some link, but subject to the spatial constraint

Table 2: Expressions of $\boldsymbol{T}_z(q_i)$ and ${}^{i-1}\boldsymbol{S}_i$, where $s_{q_i} = \sin q_i$ and $c_{q_i} = \cos q_i$

| Joint type | $\boldsymbol{T}_Z(q_i)$ | ${}^{i-1}\boldsymbol{S}_i$ |
|---|---|---|
|  | $\begin{bmatrix} c_{q_i} & -s_{q_i} & 0 & 0 \\ s_{q_i} & c_{q_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ |
|  | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ |

forces $\hat{\boldsymbol{f}}$ and $-\hat{\boldsymbol{f}}$ that the half links exert on each other (Fig 17). Geometrically, the half links play the same role as the base and tip links defined in Appendix B, but each of them has half the inertia of the original link. The force $\hat{\boldsymbol{f}}$, moreover, is a 6-D vector encoding the resultant torque and force applied by the base link on the tip link.

Under the previous view, the equation of motion of the robot takes the form

$$\boldsymbol{H}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} = \boldsymbol{Q} - \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{Q}_f, \tag{53}$$

where $\boldsymbol{H}(\boldsymbol{q})$ is the mass matrix of the system, $\boldsymbol{Q}$ is the generalised actuation force, $-\boldsymbol{C}$ is the generalised force modelling gravity and inertial velocity-product terms, and $\boldsymbol{Q}_f$ is the net generalised force of $\hat{\boldsymbol{f}}$ and $-\hat{\boldsymbol{f}}$.

Since $\boldsymbol{H}$ and $\boldsymbol{C}$ depend only on the inertial properties of the bodies and on the robot state, they can be computed by treating the system as a kinematic tree: the one that results from cutting the link as in Fig. 17, but with $\hat{\boldsymbol{f}}$ and $-\hat{\boldsymbol{f}}$ omitted. The equation of motion of such a tree is thus Eq. (53) with $\boldsymbol{Q}_f$ suppressed:

$$\boldsymbol{H}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} = \boldsymbol{Q} - \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}).$$

Note that $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{Q}$ when $\ddot{\boldsymbol{q}} = \boldsymbol{0}$, which shows that $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ can be regarded as the generalised force that imparts a null acceleration to the tree. Thus, we can easily compute $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ by setting $\ddot{\boldsymbol{q}} = \boldsymbol{0}$ in the recursive Newton-Euler method of inverse dynamics [46]. The $\boldsymbol{H}$ matrix could also be computed via inverse dynamics, but the composite rigid-body method by Walker and Orin is more efficient [68]. In our implementation, however, we compute $\boldsymbol{C}$ and $\boldsymbol{H}$ with the spatial versions of such algorithms [23].

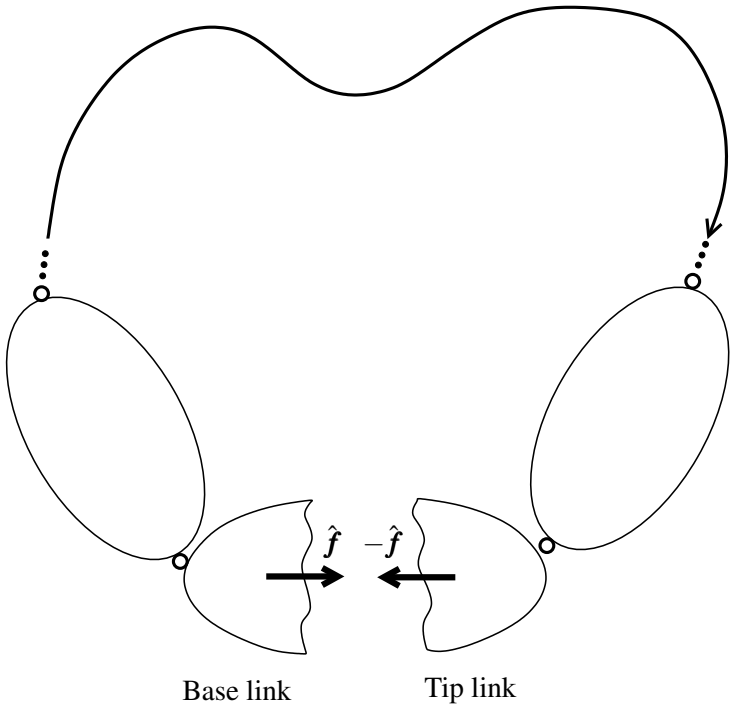



Figure 17: A kinematic loop can be thought of as being cut at some link, but subject to the spatial constraint forces $\hat{f}$ and $-\hat{f}$.

Both $\boldsymbol{Q}$ and $\boldsymbol{Q}_f$ are easy to obtain in Eq. (53). The $k$th component of $\boldsymbol{Q}$ is the motor force or torque $\tau_k$ applied at joint $k$, which is null if the joint is passive. Together, the $\tau_k$ values of the powered joints form the action vector $\boldsymbol{u}$ in Eq. (6). Moreover, it can be shown that

$$\boldsymbol{Q}_f = \boldsymbol{J}^\top \hat{\boldsymbol{f}},$$

where $\boldsymbol{J}$ is the loop Jacobian defined in Appendix B [23].

When $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\boldsymbol{u}$ are known, Eq. (53) is an undetermined system of $n$ equations in $n+6$ unknowns (the $n$ components of $\ddot{\boldsymbol{q}}$ and the 6 components of $\hat{\boldsymbol{f}}$). To be able to solve for $\ddot{\boldsymbol{q}}$, we have to supplement Eq. (53) with the time derivative of Eq. (52),

$$\boldsymbol{J}\,\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\,\dot{\boldsymbol{q}} = \boldsymbol{0}, \tag{54}$$

which adds 6 additional constraints. Together, Eqs. (53) and (54) give rise to the system

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{J}^\top \\ \boldsymbol{J} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q} - \boldsymbol{C} \\ -\dot{\boldsymbol{J}}\,\dot{\boldsymbol{q}} \end{bmatrix} \tag{55}$$

where $\boldsymbol{\lambda} = -\hat{\boldsymbol{f}} \in \mathbb{R}^6$ plays the role of a Lagrange multiplier vector. This system can be solved for $\ddot{\boldsymbol{q}}$ and $\hat{\boldsymbol{f}}$ in general, since $\boldsymbol{H}$ is positive-definite and $\boldsymbol{J}$ is full rank when the mechanism is free from C-space singularities.

It is worth noting that, while the calculation of $\dot{\boldsymbol{J}}\,\dot{\boldsymbol{q}}$ is often expensive, this term is equal to the difference between the spatial accelerations of the tip and base links when $\ddot{\boldsymbol{q}} = \boldsymbol{0}$ [23]. These accelerations are recursively obtained during the earlier calculation of $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, as part of the Newton-Euler method of inverse dynamics. Once obtained, therefore, they can be stored to construct $\dot{\boldsymbol{J}}\,\dot{\boldsymbol{q}}$ in Eq. (55).

Eq. (6) can finally be written as

$$\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \end{bmatrix}, \tag{56}$$

where

$$\boldsymbol{f}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{u}) = \begin{bmatrix}\boldsymbol{I}_n & \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{H} & \boldsymbol{J}^\top \\ \boldsymbol{J} & \boldsymbol{0}\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{Q}-\boldsymbol{C} \\ -\dot{\boldsymbol{J}}\,\dot{\boldsymbol{q}}\end{bmatrix}. \tag{57}$$

## References


[1] 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*, 21(5):834–849, 2005.

[2] D. Berenson, S. Srinivasa, and J. J. Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *International Journal of Robotics Research*, 30(12):1435–1460, 2011.

[3] D. P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, 2005.

[4] J. T. Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM Publications, 2010.

[5] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998.

[6] 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*, 200(13-16):1568–1576, 2011.

[7] James E Bobrow. Optimal robot path planning using the minimum-time criterion. *IEEE Journal on Robotics and Automation*, 4(4):443–450, 1988.

[8] James E Bobrow, Steven Dubowsky, and JS Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17, 1985.

[9] O. Bohigas, M. E. Henderson, L. Ros, M. Manubens, and J. M. Porta. Planning singularity-free paths on closed-chain manipulators. *IEEE Transactions on Robotics*, 29(4):888–898, 2013.

[10] O. Bohigas, M. Manubens, and L. Ros. *Singularities of robot mechanisms: numerical computation and avoidance path planning*, volume 41 of *Mechanisms and Machine Science*. Springer, 2017.

[11] Ilian Bonev. Delta parallel robot - the story of success. *Newsletter, available at http://www.parallemic.org*, 2001.

[12] R. Bordalba, J. M. Porta, and L. Ros. Randomized planning of dynamic motions avoiding forward singularities. In *Advances in Robot Kinematics*, pages 170–178, 2018.

[13] R. Bordalba, J. M. Porta, and L. Ros. A singularity-robust LQR controller for parallel robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 270–276, 2018.

[14] R. Bordalba, L. Ros, and J. M. Porta. Randomized kinodynamic planning for constrained systems. In *IEEE International Conference on Robotics and Automation*, pages 7079–7086, 2018.

[15] 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*, 20(2):740–749, 2015.

[16] S. Briot and V. Arakelian. Optimal force generation in parallel manipulators for passing through the singular positions. *The International Journal of Robotics Research*, 27(8):967–983, 2008.

[17] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of robot motion: theory, algorithms, and implementations*. Intelligent Robotics and Autonomous Agents. MIT Press, 2005.

[18] J. Cortés, T. Siméon, and J.-P. Laumond. A random loop generator for planning the motions of closed kinematic chains using PRM methods. In *IEEE International Conference on Robotics and Automation*, pages 2141–2146, 2002.

[19] Javier García de Jalón and Eduardo Bayo. *Kinematic and dynamic simulation of multibody systems*. Springer Verlag, 1993.

[20] 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 *2011 IEEE International Conference on Robotics and Automation*, pages 2178–2183, 2011.

[21] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993.

[22] J. Duffy. *Statics and kinematics with applications to robotics*. Cambridge University Press, 1996.

[23] R. Featherstone. *Robot dynamics algorithms*. Kluwer, Norwell, MA, 1987.

[24] R. Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[25] Roy Featherstone and David E Orin. Dynamics. In *Springer Handbook of Robotics*, pages 37–66. Springer, 2016.

[26] Siyuan Feng, Eric Whitman, X Xinjilefu, and Christopher G Atkeson. Optimization based full body control for the atlas robot. In *IEEE-RAS International Conference on Humanoid Robots*, pages 120–127, 2014.

[27] 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*, pages 2429–2436, 2013.

[28] E. Hairer. Geometric integration of ordinary differential equations on manifolds. *BIT Numerical Mathematics*, 41(5):996–1007, 2001.

[29] L. Han and N. M. Amato. A kinematics-based probabilistic roadmap method for closed chain systems. In *Algorithmic and Computational Robotics - New Directions*, pages 233–246, 2000.

[30] K. Hauser. Fast interpolation and time-optimization with contact. *The International Journal of Robotics Research*, 33(9):1231–1250, 2014.

[31] Kris Hauser and Yilun Zhou. Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space. *IEEE Transactions on Robotics*, 32(6):1431–1443, 2016.

[32] M. E. Henderson. Multiple parameter continuation: computing implicitly defined k-manifolds. *International Journal of Bifurcation and Chaos*, 12(3):451–476, 2002.

[33] Robert P Hoyt. Spiderfab: An architecture for self-fabricating space systems. In *AIAA Space 2013 Conference and Exposition*, page 5509, 2013.

[34] L. Jaillet and J. M. Porta. Path planning under kinematic constraints by rapidly exploring manifolds. *IEEE Transactions on Robotics*, 29(1):105–117, 2013.

[35] 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*, pages 4569–4574, 2011.

[36] B. Kim, T. T. Um, C. Suh, and F. C. Park. Tangent bundle RRT: A randomized algorithm for constrained motion planning. *Robotica*, 34(1):202–225, 2016.

[37] Zachary Kingston, Mark Moll, and Lydia E Kavraki. Decoupling constraints from sampling-based planners. In *International Symposium of Robotics Research*, 2017.

[38] Zachary Kingston, Mark Moll, and Lydia E Kavraki. Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:159–185, 2018.

[39] 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*, 4(3):277–283, 2019.

[40] 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*, pages 3713–3719, 2014.

[41] S. M. LaValle. *Planning algorithms*. Cambridge University Press, New York, 2006.

[42] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[43] Sung-Hee Lee, Junggon Kim, Frank Chongwoo Park, Munsang Kim, and James E Bobrow. Newton-type algorithms for dynamics-based robot movement optimization. *IEEE Transactions on robotics*, 21(4):657–667, 2005.

[44] F. L. Lewis, D. Vrabie, and V. L. Syrmos. *Optimal control*. John Wiley & Sons, 2012.

[45] Y. Li, Z. Littlefield, and K. E. Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[46] John YS Luh, Michael W Walker, and Richard PC Paul. On-line computational scheme for mechanical manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102(2):69–76, 1980.

[47] 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*, pages 2537–2542, 2012.

[48] L. R. Petzold. Numerical solution of differential-algebraic equations in mechanical systems simulation. *Physica D: Nonlinear Phenomena*, 60(1–4):269–279, 1992.

[49] Friedrich Pfeiffer and Rainer Johanni. A concept for manipulator trajectory planning. *IEEE Journal on Robotics and Automation*, 3(2):115–123, 1987.

[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*, 36(1):44–67, 2017.

[51] Q.-C. Pham, S. Caron, and Y. Nakamura. Kinodynamic planning in the configuration space via admissible velocity propagation. In *Robotics: Science and Systems*, 2013.

[52] Quang-Cuong Pham. A general, fast, and robust implementation of the time-optimal path parameterization algorithm. *IEEE Transactions on Robotics*, 30(6):1533–1540, 2014.

[53] Quang-Cuong Pham and Olivier Stasse. Time-optimal path parameterization for redundantly actuated robots: A numerical integration approach. *IEEE/ASME Transactions on Mechatronics*, 20(6):3257–3263, 2015.

[54] J. M. Porta, L. Jaillet, and O. Bohigas. Randomized path planning on manifolds based on higher-dimensional continuation. *The International Journal of Robotics Research*, 31(2):201–215, 2012.

[55] 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*, 21(3):105–114, 2014.

[56] M. Posa, S. Kuindersma, and R. Tedrake. Optimization and stabilization of trajectories for constrained dynamical systems. In *IEEE International Conference on Robotics and Automation*, pages 1366–1373, 2016.

[57] F. A. Potra and J. Yen. Implicit numerical integration for Euler-Lagrange equations via tangent space parametrization. *Journal of Structural Mechanics*, 19(1):77–98, 1991.

[58] J.-P. Samin and P. Fisette. *Symbolic modeling of multibody systems*. Springer, 2003.

[59] J. Schulman, Duan 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*, 33(9):1251–1270, 2014.

[60] Z. Shiller and S. Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics and Automation*, 7(6):785–797, Dec 1991.

[61] Zvi Shiller and Steven Dubowsky. Robot path planning with obstacles, actuator, gripper, and payload constraints. *The International Journal of Robotics Research*, 8(6):3–18, 1989.

[62] Zvi Shiller and Hsueh-Hen Lu. Computation of path constrained time optimal motions with dynamic singularities. *Journal of Dynamic Systems, Measurement, and Control*, 114(1):34–40, 1992.

[63] Kang Shin and Neil McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, 30(6):531–541, 1985.

[64] J-JE Slotine and Hyun S Yang. Improving the efficiency of time-optimal path-following algorithms. *IEEE Transactions on Robotics and Automation*, 5(1):118–124, 1989.

[65] M. Stilman. Task constrained motion planning in robot joint space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3074–3081, 2007.

[66] Russ Tedrake. LQR-Trees: Feedback motion planning on sparse randomized trees. In *Robotics: Science and Systems*, 2009.

[67] L.-W. Tsai. *Robot analysis: The mechanics of serial and parallel manipulators*. John Wiley and Sons, 1999.

[68] Michael W Walker and David E Orin. Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, 104(3):205–211, 1982.

[69] 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*, pages 5054–5061, 2013.

[70] 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*, 17(6):951–958, Dec 2001.

[71] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. CHOMP: Covariant Hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.

## IRI reports

This report is in the series of IRI technical reports.
All IRI technical reports are available for download at the IRI website
`http://www.iri.upc.edu`.