A Randomised Kinodynamic Planner for Closed-chain Robotic Systems

Ricard Bordalba, Lluís Ros, and Josep M. Porta

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, collaborative 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 that complicates the design of the sampling and steering procedures, and leads to trajectories that drift from the manifold if standard integration methods are used. To address these issues, this paper presents a kinodynamic RRT planner that constructs an atlas of the state space incrementally, and uses this atlas to generate random states, and to dynamically steer the system towards such states. The steering method exploits the atlas charts to compute locally optimal policies based on linear quadratic regulators. The atlas also allows the integration of the equations of motion as a differential equation on the manifold, which eliminates any drift from such manifold and 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 planner performance 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.**

Index terms—**Kinodynamic motion planning, loop-closure constraint, closed kinematic chain, atlas, manifold, LQR, steering.**

I. Introduction

SINCE its formalization in the early nineties [1], the kinodynamic planning problem remains as one of the most challenging open problems in robotics. The problem entails finding feasible trajectories connecting given states of a robot, each defined by a configuration and a velocity of the underlying mechanical system. 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 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.

This work has been partially funded by the Spanish Ministry of Science, Innovation, and Universities under project DPI2017-88282-P

Ricard Bordalba, Lluís Ros, and Josep M. Porta are with the Institut de Robòtica i Informàtica Industrial (CSIC-UPC). Llorens Artigas 4-6, 08028 Barcelona, Catalonia. E-mails: rbordalba@iri.upc.edu, ros@iri.upc.edu, and porta@iri.upc.edu.

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 [2], as they tend to exploit 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. Moreover, the obstacle region is virtually larger, involving states that correspond to an actual collision, but also those from which a future collision is inevitable due to the system momentum. The planning of steering motions is considerably more difficult as well. While direct motions suffice in the C-space, steering motions in the state space need to conform to the vector fields defined by the equations of motion and to the actuator limits of the robot.

Among all kinodynamic planning techniques, rapidly-exploring random trees (RRTs) have emerged as one of the most successful planning paradigms to date [7]. RRTs make 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 technique is fairly general and, with proper extensions, can even converge to minimum-cost motions [8, 9]. However, existing RRT methods also suffer from a main limitation: they assume that the robot state can be described by means of independent generalised coordinates. This makes them applicable to open-chain robots, or to robots with explicit state space parametrisations, but they have problems in dealing with general mechanisms with 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 recent literature [10]–[16].

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

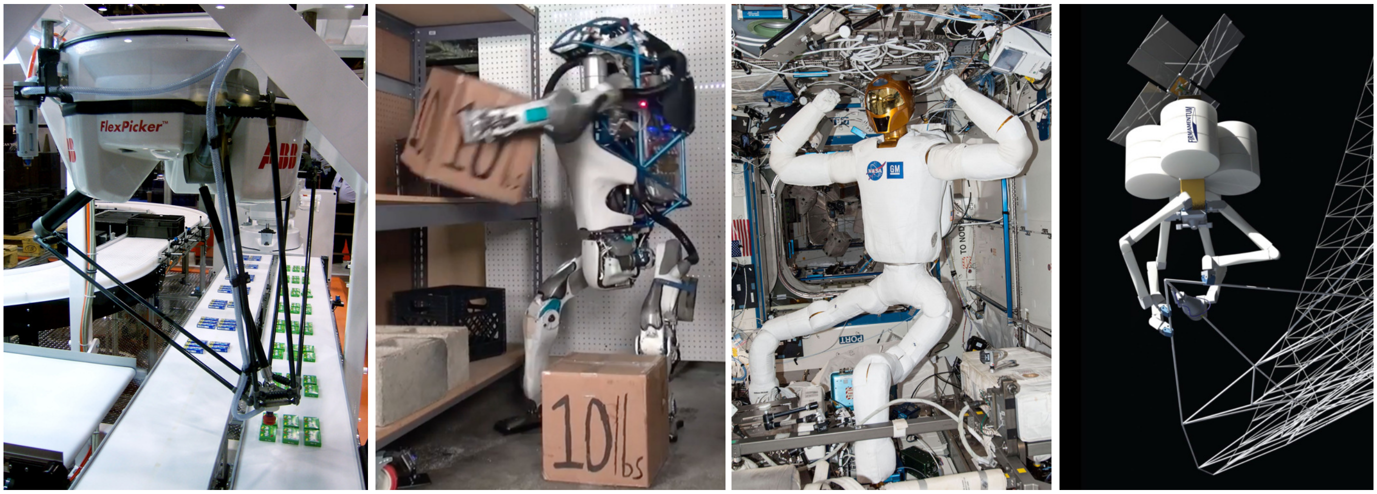


Fig. 1. Example systems involving closed kinematic chains. The chains may be intrinsic to the robot structure, as in parallel robots (left picture), or they may result from manipulation constraints during a task, as in multi-limb systems transporting an object, or keeping feet attached to the environment (right pictures). From left to right: A Delta parallel robot [3], the Atlas robot from Boston Dynamics lifting a heavy load [4], the Robonaut 2 robot with two legs clamped to the International Space Station [5], and the SpiderFab Bot, a conceptual design for self-fabricating space systems [6]. Pictures courtesy of ABB, Boston Dynamics, NASA, and Tethers Unlimited, Inc (respectively).

robot is not properly handled, the planned trajectories may deviate substantially from the manifold, leading to undesired violations of the kinematic constraints, or to failure to reach the goal. Forward singularities may also complicate the planning and control of motions across certain surfaces of the state space [17].

The purpose of this paper is to extend the planner in [7] 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 feasible trajectories even across forward singularities. An early version of our planner was presented in [18]. In contrast to [18], we here develop a steering method based on linear quadratic regulators (LQR), which greatly increases the planner efficiency in comparison to the randomised strategy used in [18]. New challenging test cases are also reported 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 [18]. It is worth noting that, while some path planning approaches have previously dealt with closed kinematic chains [10]–[12, 15, 19]–[23], none of them 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 [12] to cope with dynamic constraints.

The rest of the paper is organised as follows. Section II reviews the state of the art on kinodynamic planning to better place our work into context. Section III formally states the problem we confront, enumerating our assumptions and the various constraints intervening. Section IV explains why most RRT approaches, while powerful, are limited in some way or another, and would be difficult to extend to cope with closed chains and dynamic constraints simultaneously. Sections V and VI present effective sampling, simulation, and steering methods that allow us to describe, in Section VII, our planner implementation. Sections VIII and IX respectively examine the completeness properties of the planner and its practical performance in illustrative situations. Section X finally provides the paper conclusions and discusses several points requiring further attention.

II. Related work

A. C-space approaches

The sheer complexity of kinodynamic planning is usually managed by decomposing the problem into two simpler problems [24]. 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, like probabilistic roadmaps or randomised tree techniques among others [24, 25], the trajectory is usually obtained with the time-scaling method in [26] or its later improvements [27]–[30]. 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 obtained [31].

The previous approach generates a trajectory that is only time-optimal for the computed path, but makes the problem more tractable, so it can be solved in systems with many degrees of freedom like humanoids, legged robots, or mobile robot formations [32]. Its lack of completeness, moreover, can be alleviated by improving the trajectory a posteriori using optimization techniques [33]–[35]. 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 [36], which can be combined with randomised planners to generate graceful dynamic motions [32].

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, e.g., 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 [13, 14], parallel robots with passive joints are underactuated at forward singularities [17]. These configurations are problematic when managed in the C-space as they can only be traversed under particular velocities and accelerations. As it will turn out, however, the previous limitations do not apply if robot trajectories are directly planned in the state space.

B. State space approaches

Existing techniques for planning in the state space can roughly be grouped into optimization and randomised approaches. On the one hand, optimization approaches can be applied to remarkably-complex problems [37]–[41]. 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 needed. However, there is a trade-off between the number of knot points, the integration method adopted, and the computational cost of the optimization problem. In systems with closed kinematic chains, moreover, the discretization of the differential equations produces knot points that easily drift from the state space manifold, which results in unwanted link disassemblies or contact losses. In [38], differential constraints were approximated explicitly by means of polynomial functions while guaranteeing third-order integration accuracy in constrained systems. Even so, the problem size becomes huge for long time horizons or systems with many degrees of freedom [32]. Good discussions on the advantages and pitfalls of optimization-based techniques can be found in [42] and [13]. On the other hand, randomised approaches like the standard RRT [7] can cope with differential constraints in relatively high-dimensional problems, and guarantee to find a solution when it exists and enough computing time is available. A main issue, however, is that exact steering methods are not available for nonlinear dynamical systems. The usual RRT method tries to circumvent this problem by simulating random actions for a given time, and then selecting the action that gets the system closest to the target [7]. For particular systems, better solutions exist though. For instance, the approach in [43] assumes double integrator dynamics and exploits the fact that the minimum time problem has an efficient solution in this case. The resulting planner is fast, but the full dynamics of the system can only be coped via feedback linearisation, which requires the inverse dynamic problem to be solvable. The method in [44] linearises the system dynamics and uses an infinite-horizon LQR controller to define a steering method, but such a controller can only be used to reach zero-velocity states. In contrast, [45], [46], and [47] use finite-horizon LQR controllers that that can converge to arbitrary states. 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 the one in [47], but extended to cope with such chains.

III. Problem formulation

To formally state our problem, let us describe the robot configuration by means of a tuple $\boldsymbol{q}$ of n_q generalised 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

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

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

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

which may be quite complex in general. In this paper, 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 $\mathcal{C}$ is a smooth manifold of dimension $d_{\mathcal{C}} = n_q - n_e$ without C-space singularities [17]. This assumption is not too restrictive, as these singularities are often removed by mechanical designers (e.g., by setting appropriate joint limits), and it does not rule out forward or inverse singularities [17], which can be crossed naturally by our planner.

By differentiating Eq. (1) with respect to time, we obtain the velocity equation of the robot

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

which characterises the feasible vectors $\dot{\boldsymbol{q}}$ at a given $\boldsymbol{q} \in \mathcal{C}$.

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

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

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 $\boldsymbol{x}$,

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

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

We 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

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

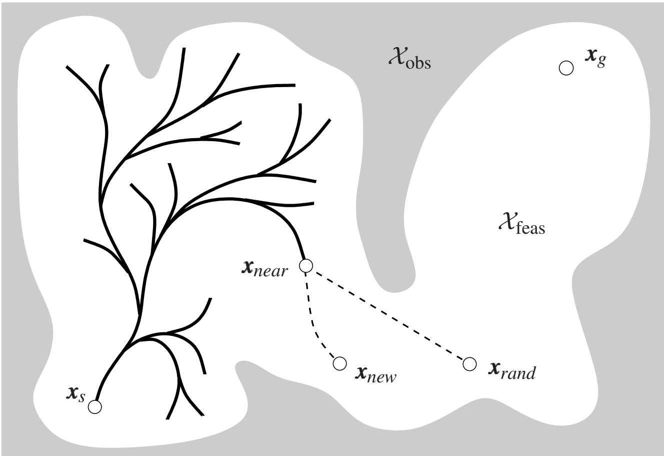


Fig. 2. Expansion of a unidirectional RRT [7].

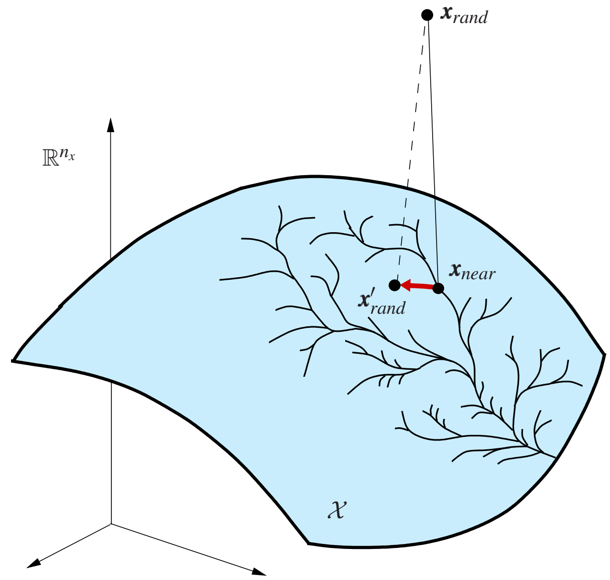


Fig. 3. Generation of a guiding sample according to [24].

In this system, Eq. (5) forces the states $\boldsymbol{x}$ to remain in $\mathcal{X}$, and Eq. (6) models the dynamics of the robot, which can be described using the multiplier form of the Euler-Lagrange equations for example [48]. For each value of $\boldsymbol{u}$, Eq. (6) defines a vector field over $\mathcal{X}$, which can be used together with Eq. (5) to simulate the robot motion forward in time using proper integration tools [49].

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 of the i-th motor. 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 problem we confront can be stated 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 $\boldsymbol{u} = \boldsymbol{u}(t)$ lying in $\mathcal{U}$ for all t such that the trajectory $\boldsymbol{x} = \boldsymbol{x}(t)$ determined by Eqs. (5) and (6) for $\boldsymbol{x}(0) = \boldsymbol{x}_s$ fulfils $\boldsymbol{x}(t_g) = \boldsymbol{x}_g$ at some time $t_g > 0$, with $\boldsymbol{x}(t) \in \mathcal{X}_{\text{feas}}$ for all $t \in [0, t_g]$.

IV. Limitations of prior RRT methods

Observe that the previous formulation is more general than the one assumed in earlier RRT planners. In particular, the approaches in [10, 12, 19, 22, 23] are purely kinematic, so they only consider Eq. (1), and neglect Eqs. (2), (6), and the force bounds in (7). As a result, they only compute paths in $\mathcal{C}$, and such paths might be unfeasible dynamically. In contrast, kinodynamic approaches like [7]–[9, 44, 46, 47] consider Eq. (6) and the bounds in (7), but not Eqs. (1) and (2), which impedes the handling of robots with closed kinematic chains. While [24] proposed a few extensions to help RRTs cope with such chains, we next see that these lead to unsatisfactory results.

Recall from [24] that a usual RRT is initialised at $\boldsymbol{x}_s$, and is extended by applying four 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 Δt 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 former case, $\boldsymbol{u}$ is stored in the new edge connecting $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{new}$. The process terminates when a tree node is close enough to $\boldsymbol{x}_g$. It is worth noting that, in many implementations, steps 3) and 4) are repeated with $\boldsymbol{x}_{new}$ playing the role of $\boldsymbol{x}_{near}$, as long as $\boldsymbol{x}_{new}$ gets closer to $\boldsymbol{x}_{rand}$.

Three problems arise when applying the previous method to closed kinematic chains. First, the points $\boldsymbol{x}_{rand}$ are difficult to obtain in general, as $\mathcal{X}$ may be a manifold without explicit parametrizations. To circumvent this issue, [24, 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. 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 exploration bias of the RRT. This effect was analysed in [12] and [23]. A second problem concerns the dynamic simulation of robot motions. Existing RRT methods would only use Eq. (6) to generate such motions on the grounds that Eq. (5) is implicitly accounted for by Eq. (6) [24, Sec. 13.4.3.1]. However, from multibody dynamics it is known that the motion of a closed-chain robot can only be predicted reliably if Eq. (5) is actively used during the integration of Eq. (6) [50]. Otherwise, the inevitable errors introduced when discretising Eq. (6) will make the trajectory $\boldsymbol{x}(t)$ increasingly drift from $\mathcal{X}$ as the simulation progresses. Such a drift may even be large enough to prevent the connection of $\boldsymbol{x}_s$ with $\boldsymbol{x}_g$ [18]. The use of Baumgarte stabilization to compensate this drift [51] is also problematic, as it may lead to instabilities [52] or fictitious energy increments, and its stabilising parameters are not easy to tune. A third problem, finally, concerns the steering method. A shooting strategy based on simulating random actions from $\mathcal{U}$ was proposed in [7], but this technique is inefficient when n_u is large, as the number of samples needed to properly represent $\mathcal{U}$ grows exponentially with n_u. The lack of a good steering strategy is a general problem of RRT methods, but it is more difficult to address when closed kinematic chains are present.

Purely kinematic planners like [10, 12, 19, 22, 23] do not perform dynamic simulations, and employ direct steering motions between configurations. Moreover, most of them sample in ambient space. Thus, the previous three problems would also arise when trying to generalise them to cope with dynamic constraints. Among such planners, however, the one in [12] is more amenable for generalisation, as it employs atlas machinery that is applicable to mechanisms of general loop topology. Our goal in this paper is to show that, precisely, such a machinery can be extended to cope with the more general problem of Section III. As we shall see, once an atlas of $\mathcal{X}$ is obtained, we will have the necessary tools to 1) sample the $\mathcal{X}$ manifold directly instead of its ambient space $\mathbb{R}^{n_\mathcal{X}}$; 2) integrate Eqs. (5) and (6) as a true differential-algebraic equation so as to ensure driftless motions on $\mathcal{X}$; and 3) define a proper steering method for closed kinematic chains. We develop these tools in the following two sections, and later use them as basic building blocks in our planner implementation.

V. Mapping and exploring the state space

A. 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 [53] to define them for any manifold. Under 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

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

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

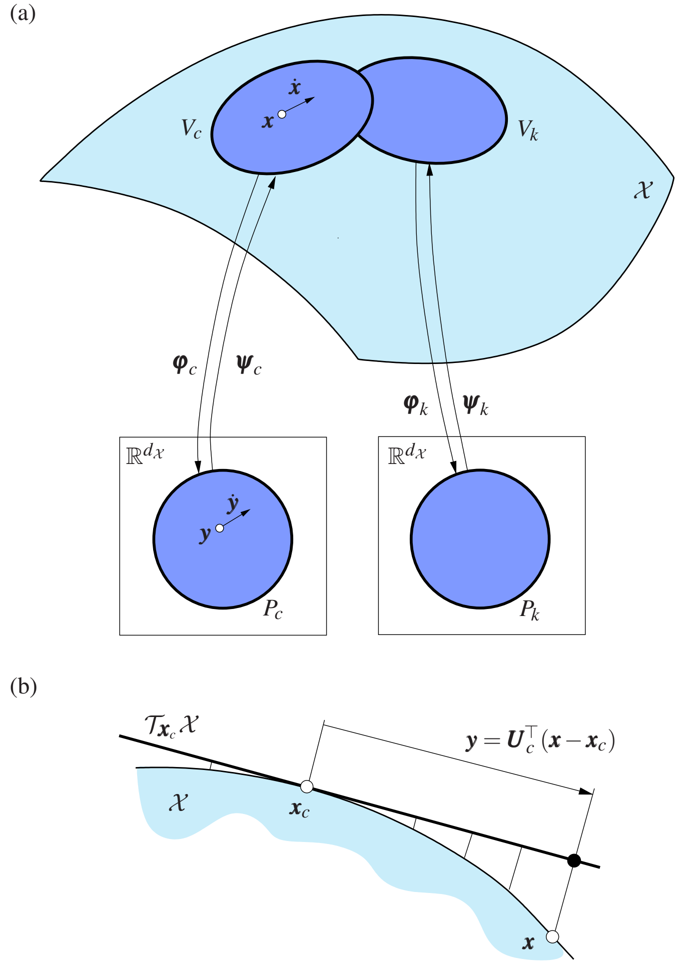


Fig. 4. (a) Two neighbouring charts of $\mathcal{X}$, labelled c and k, together with their maps $\boldsymbol{\varphi}_c$ and $\boldsymbol{\varphi}_k$, and inverse maps $\boldsymbol{\psi}_c$ and $\boldsymbol{\psi}_k$. (b) Using the tangent space parametrisation, $\boldsymbol{\varphi}_c$ is defined by the projection of $\boldsymbol{x}$ onto $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$.

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 on the P_c sets of the charts. 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 [54, 55]. 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).

B. Incremental atlas and RRT expansion

One could use the methods in [53] to construct a full atlas of the implicitly-defined state space and then use its local

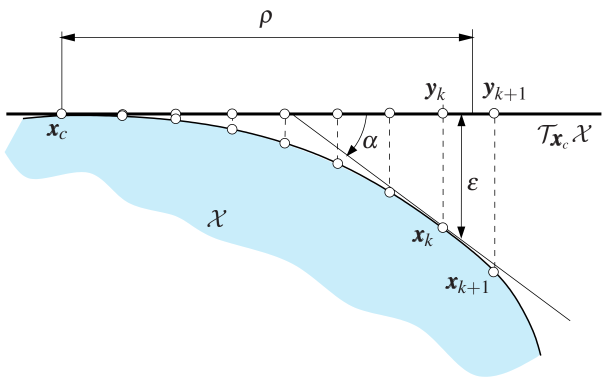


Fig. 5. Thresholds determining the extension of the P_c set of the chart at $\boldsymbol{x}_c$. While $\boldsymbol{y}_k$ lies in P_c, $\boldsymbol{y}_{k+1}$ does not because it violates Eqs. (11)-(13).

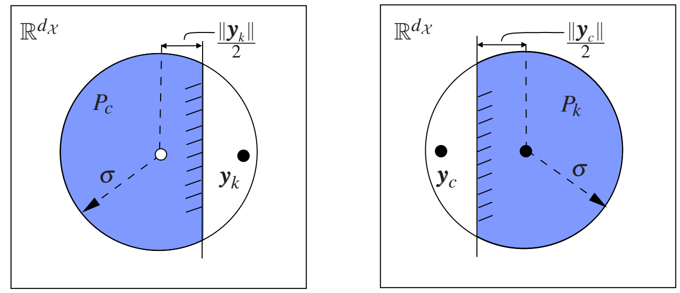


Fig. 6. Half planes added to trim the P_c and P_k sets of two neighboring charts. Note that $\boldsymbol{y}_k = \boldsymbol{\varphi}_c(\boldsymbol{x}_k)$ and $\boldsymbol{y}_c = \boldsymbol{\varphi}_k(\boldsymbol{x}_c)$.

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. For these reasons, as in [12], we combine the construction of the atlas and the expansion of the RRT. In this approach, a partial atlas is used to both generate random states and 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 $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ are two consecutive states along an RRT branch and let $\boldsymbol{y}_k$ and $\boldsymbol{y}_{k+1}$ be their local coordinate vectors in $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$. Then, a new chart at $\boldsymbol{x}_k$ is created if Eq. (9) cannot be solved for $\boldsymbol{x}_{k+1}$ using the Newton-Raphson method, or if any of the following conditions is met

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

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

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

where ε, α, and ρ 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 $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$ and the manifold $\mathcal{X}$. The second condition ensures a bounded curvature in the part of $\mathcal{X}$ that is covered by a chart, as well as a smooth transition between neighbouring charts. The third condition finally guarantees the generation of new charts as the RRT grows, even for almost flat manifolds.

C. Chart coordination

Since the charts will be used to generate samples on $\mathcal{X}$, it is important to reduce the overlap between new charts and those already present in the atlas. Otherwise, the areas of $\mathcal{X}$ covered by several charts would be oversampled. To avoid this problem, the P_c set of each chart is initialised as a ball of radius σ 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 σ of the initial ball must be larger than ρ to guarantee that the RRT branches covered by chart c will eventually trigger the generation of new charts, i.e., to guarantee that Eq. (13) will eventually hold. Also, since Eq. (13) forces the norm of $\boldsymbol{y}_k$ to be limited by ρ, the half-plane defined by Eq. (14) will be guaranteed to clip P_c. Consequently, the P_c sets of those charts surrounded by neighbouring charts will be significantly smaller than the P_c sets of the charts at the exploration border of the atlas. As we shall see in Section VII-A, this favours the growth of the tree towards unexplored regions of $\mathcal{X}$.

VI. A STEERING METHOD

Our planner can adopt different strategies to steer the system from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{rand}$. A simple one, called randomised steering, consists in simulating several random actions in $\mathcal{U}$, and then choosing the one that brings the robot closest to $\boldsymbol{x}_{rand}$. This strategy was proposed in [7] and is the one we adopted in our early version of the planner [18]. As explained in Section IV, however, this approach becomes inefficient as the dimension of $\mathcal{U}$ increases. To amend this problem we next propose another strategy, called LQR steering, based on linear quadratic regulators. While LQR techniques are classical steering methods for control systems [56], they assume the state coordinates to be independent, so they are applicable to open chain robots only. However, we next show that, using the atlas charts, they can be extended to the closed chain case. The idea is to exploit system linearisations at the various chart centres so as to obtain a sequence of control laws bringing the robot from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{rand}$.

A. System linearisation at a chart centre

To apply LQR techniques to our steering 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, note that we cannot linearise Eq. (6), 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 $\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{y}}\big|_{\boldsymbol{y}=\boldsymbol{0}} = \boldsymbol{U}_c$ [48], the $\boldsymbol{A}$ and $\boldsymbol{B}$ terms can be written as:

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

and

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

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

B. Steering on a single chart

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

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

subject to the constraints

$$\dot{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{c}, \tag{18}$$

$$\boldsymbol{y}(0) = \boldsymbol{y}_{near}, \tag{19}$$

$$\boldsymbol{y}(t_f) = \boldsymbol{y}_{rand}. \tag{20}$$

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

The problem just formulated is known as the fixed final state optimal control problem [56]. We shall solve this problem in two stages. Initially, we will obtain $\boldsymbol{u}^*(t)$ assuming that t_f is fixed, and then we will find a time t_f that leads to a minimum of $J(\boldsymbol{u}(t),t_f)$.

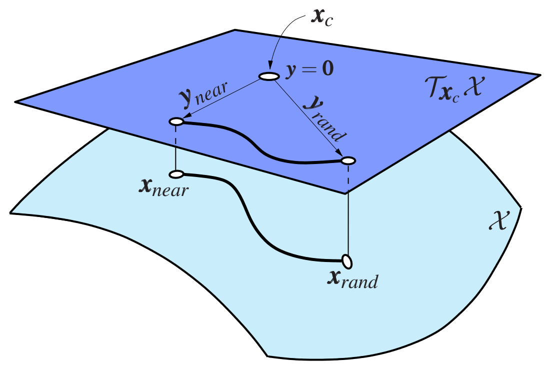


Fig. 7. When $\boldsymbol{x}_{near}$ and $\boldsymbol{x}_{rand}$ are covered by a same chart, the steering of the system can be reduced to a steering problem in $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$.

C. 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 [57]. 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

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

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:

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

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 $t > 0$ [47], which ensures that $\boldsymbol{G}(t_f)^{-1}$ always exists.

D. 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

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

The time t_f^* is thus the one that minimises $J(t_f)$ 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.

E. 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. V-B). 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 often 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. Even so, such situations do not prevent the connection of $\boldsymbol{x}_s$ with $\boldsymbol{x}_g$, as the steering algorithm is to be used inside a higher-level RRT planner. The implementation of such a planner is next addressed.

VII. 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 [7]. 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 initialised with one chart centred at $\boldsymbol{x}_s$ and another chart centred at $\boldsymbol{x}_g$ (line 1). As in [7], 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 $\boldsymbol{x}_{rand}$ using the Connect method (line 7). This method reaches a state $\boldsymbol{x}_{new}$ that, due to the presence of obstacles or to a failure of the steering procedure, may be different from $\boldsymbol{x}_{rand}$. Then, the state in T_g that is nearest to $\boldsymbol{x}_{new}$ is determined ($\boldsymbol{x}'_{near}$ in line 8) and T_g is extended from $\boldsymbol{x}'_{near}$ with the aim of reaching $\boldsymbol{x}_{new}$ (line 9). This extension generates a new state $\boldsymbol{x}'_{new}$. After this step, the trees are swapped (line 10) and, if the last connection was unsuccessful, i.e., if $\boldsymbol{x}_{new}$ and $\boldsymbol{x}'_{new}$ are not closer than a user-provided threshold (line 11), lines 5 to 10 are repeated again. If the connection was successful, a solution trajectory is reconstructed using the paths from $\boldsymbol{x}_{new}$ and $\boldsymbol{x}'_{new}$ to the roots of T_s and T_g (line 12). Different metrics

Algorithm 1: The top-level pseudocode of the planner

```
PLAN TRAJECTORY(x_s, x_g)
input : The query states, x_s and x_g.
output: A trajectory connecting x_s and x_g.
1  A ← INITATLAS(x_s, x_g)
2  T_s ← INITRRT(x_s)
3  T_g ← INITRRT(x_g)
4  repeat
5  |   x_rand ← SAMPLE(A, T_s)
6  |   x_near ← NEARESTSTATE(T_s, x_rand)
7  |   x_new ← CONNECT(A, T_s, x_near, x_rand)
8  |   x'_near ← NEARESTSTATE(T_g, x_new)
9  |   x'_new ← CONNECT(A, T_g, x'_near, x_new)
10 |   SWAP(T_s, T_g)
11 until ||x_new − x'_new|| < β
12 RETURN(TRAJECTORY(T_s, x_new, T_g, x'_new))
```

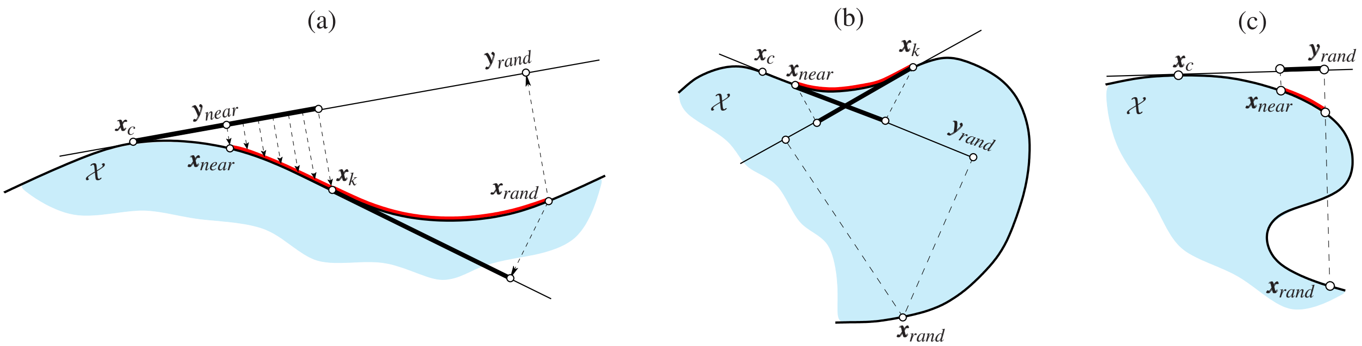


Fig. 8. (a) Steering towards states not covered by the chart of $\boldsymbol{x}_{near}$. (b) Cyclic behavior of the steering method. (c) Convergence to $\boldsymbol{y}_{rand}$ but not to $\boldsymbol{x}_{rand}$.

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(σ)
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 $\quad c \leftarrow$ CHARTINDEX$(\boldsymbol{x}_{near})$
5 $\quad (\boldsymbol{u}^*, t^*_f) \leftarrow$ LQRPOLICY$(A,\boldsymbol{x}_{near},\boldsymbol{x}_{rand})$
6 $\quad$ **if** $t^*_f \leq t^*_{fp}$ **then**
7 $\quad\quad t^*_{fp} \leftarrow t^*_f$
8 $\quad\quad (\boldsymbol{x}_{new},\boldsymbol{u}_{new}) \leftarrow$ SIMULATE$(A,c,\boldsymbol{x}_{near},\boldsymbol{x}_{rand},\boldsymbol{u}^*,t^*_f)$
9 $\quad\quad$ **if** $\boldsymbol{x}_{new} \in \mathcal{X}_{feas}$ **and** $\boldsymbol{x}_{new} \neq \boldsymbol{x}_{near}$ **then**
10 $\quad\quad\quad T \leftarrow$ ADDEDGE$(T,\boldsymbol{x}_{near},\boldsymbol{u}_{new},\boldsymbol{x}_{new})$
11 $\quad\quad\quad \boldsymbol{x}_{near} \leftarrow \boldsymbol{x}_{new}$
12 **until** $t^*_f > t^*_{fp}$ **or** $\|\boldsymbol{\varphi}_c(\boldsymbol{x}_{new}) - \boldsymbol{\varphi}_c(\boldsymbol{x}_{rand})\| \leq \delta$ **or** $\boldsymbol{x}_{new} \notin \mathcal{X}_{feas}$
13 $\boldsymbol{x}_{new} \leftarrow \boldsymbol{x}_{near}$
14 RETURN$(\boldsymbol{x}_{new})$

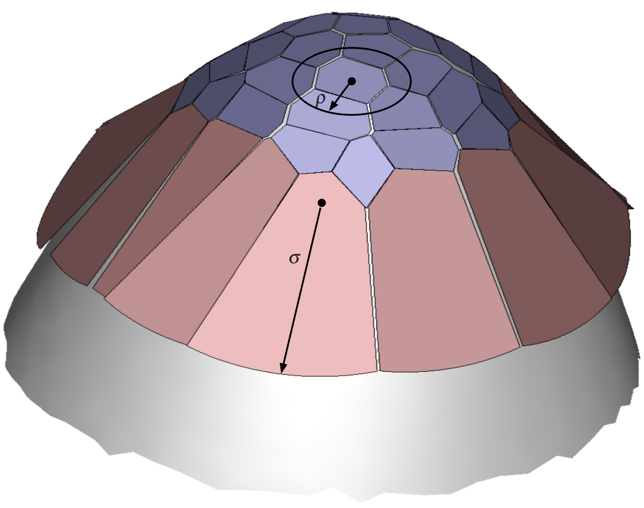


Fig. 9. A partial atlas of a paraboloid, with its inner and border charts colored in blue and red, respectively. Black dots indicate chart centers.

can be used to determine the distance between two states without affecting the overall structure of the planner. As in [7], we use Euclidean distance for simplicity.

A. 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 then randomly sampled also with uniform distribution inside a ball of radius σ centred at the origin of $\mathbb{R}^{d_\mathcal{X}}$ (line 3). Chart selection and parameter sampling are repeated until $\boldsymbol{y}_{rand}$ falls inside the P_c set for the selected chart. This process generates a sample $\boldsymbol{y}_{rand}$ with uniform distribution over the union of the P_c sets covering T. Note here that the P_c set of a chart in the interior of the atlas is included in a ball of radius ρ, while the P_c set of a chart at the border of the atlas is included inside a ball of radius $\sigma > \rho$ (Fig. 9). If we fix ρ but increase σ the overall volume of the border charts increases, whereas that of the inner charts stays constant. Therefore, by increasing σ we can increase the exploration bias of the algorithm. This bias is analogous to the Voronoi bias in standard RRTs [58]. After generating a valid sample, 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 still provides a guiding direction to steer the tree towards unexplored regions of $\mathcal{X}$.

B. Tree extension

Algorithm 3 attempts to connect a state $\boldsymbol{x}_{near}$ to a state $\boldsymbol{x}_{rand}$ using LQR steering. The analogous procedure using ran-

Algorithm 4: Simulate an action.

SIMULATE($A, c, \boldsymbol{x}_k, \boldsymbol{x}_{rand}, \boldsymbol{u}^*, t_f^*$)

input : An atlas, A, the chart index c, 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.

1 $t \leftarrow 0$
2 $\boldsymbol{u}_k \leftarrow \emptyset$
3 VALIDSTATE $\leftarrow$ TRUE
4 **while** VALIDSTATE **and** $\|\boldsymbol{\varphi}_c(\boldsymbol{x}_k) - \boldsymbol{\varphi}_c(\boldsymbol{x}_{rand})\| > \delta$ **and** $|t| < t_f^*$ **do**
5 $\quad(\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}, h) \leftarrow$ NEXTSTATE($\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{u}^*(t), \boldsymbol{F}, \boldsymbol{x}_c, \boldsymbol{U}_c, \delta$)
6 $\quad$**if** $\boldsymbol{x}_{k+1} \notin \mathcal{X}_{feas}$ **then**
7 $\quad\quad\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k+1}$
8 $\quad\quad$VALIDSTATE $\leftarrow$ FALSE
9 $\quad$**else**
10 $\quad\quad$**if** $\|\boldsymbol{x}_{k+1} - (\boldsymbol{x}_c + \boldsymbol{U}_c\, \boldsymbol{y}_{k+1})\| > \varepsilon$ **or**
11 $\quad\quad\quad\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\| / \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| < \cos(\alpha)$ **or**
12 $\quad\quad\quad\|\boldsymbol{y}_{k+1}\| > \rho$ **then**
13 $\quad\quad\quad$ADDCHARTTOATLAS($A, \boldsymbol{x}_k$)
14 $\quad\quad\quad$VALIDSTATE $\leftarrow$ FALSE
15 $\quad\quad$**else**
16 $\quad\quad\quad\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k+1}$
17 $\quad\quad\quad\boldsymbol{u}_k \leftarrow \boldsymbol{u}_k \cup \{(\boldsymbol{u}(t), h)\}$
18 $\quad\quad\quad t \leftarrow t + h$
19 $\quad\quad\quad$**if** $\boldsymbol{y}_{k+1} \notin P_c$ **then**
20 $\quad\quad\quad\quad$VALIDSTATE $\leftarrow$ FALSE
21 RETURN($\boldsymbol{x}_k, \boldsymbol{u}_k$)

domised steering is available in [18]. The algorithm implements a loop where, initially, the optimal policy $\boldsymbol{u}^*$ and time t_f^* to connect these two states are computed (line 5). 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 8). The simulation produces a state $\boldsymbol{x}_{new}$ which, if it is feasible and different from $\boldsymbol{x}_{near}$, it is added to the tree. This involves the creation of an edge between $\boldsymbol{x}_{near}$ and $\boldsymbol{x}_{new}$ (line 10), which stores the control sequence $\boldsymbol{u}_{new}$ executed in the simulation. The loop is repeated until t_f^* is larger than t_{fp}^* (line 12), or $\boldsymbol{x}_{rand}$ is reached with accuracy δ in parameter space, or the next state is unfeasible.

Algorithm 4 summarises the procedure used to simulate a given policy $\boldsymbol{u}^*(t)$ from a particular state $\boldsymbol{x}_k$. The simulation progresses while the new state is valid, the target state is not reached with accuracy δ in parameter space, and the integration time t is lower than t_f^* (line 4). A state is not valid if is not in $\mathcal{X}_{\text{feas}}$ (line 8), if it is not in the validity area of the chart (line 14), or if it is not included in the current P_c set (line 20), i.e., it is parametrised by a neighbouring chart. 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, on a newly created chart (line 13) or on the neighbouring chart, respectively.

The key procedure in the simulation is the NEXTSTATE method (line 5), which provides the next state $\boldsymbol{x}_{k+1}$, given the current state $\boldsymbol{x}_k$ and the action $\boldsymbol{u}^*(t)$ at time t. The elements of $\boldsymbol{u}^*(t)$ are saturated to their bounds in Eq. (7) if such bounds are surpassed. Then, the simulation is implemented by integrating Eq. (6) using local coordinates as explained in Section V-A. 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 [49]. 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 preferable as it avoids the computation of the Jacobian of the system at each step. Potra and Yen [49] gave an approximation of this Jacobian that allows finding $\boldsymbol{x}_{k+1}$ in only a few iterations. For backward integration, i.e., when extending the RRT with root at $\boldsymbol{x}_g$, the time step h in Eq. (36) must simply be negative.

C. Setting the planner parameters

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

Parameters ε, α, and ρ appear, respectively, in Eqs. (11), (12), and (13). Parameter α bounds the angle between neighboring charts. This angle should be small, otherwise the V_c sets for neighboring charts might not overlap, impeding a smooth transition between the charts [53]. Such problematic areas can be detected and patched [12], but this process introduces inefficiencies. Thus, we suggest to keep this parameter below $\pi/6$. Parameter ε is only relevant if the distance between the manifold and the tangent space becomes large without a significant change in curvature, which rarely occurs. Since this distance is computed in ambient space, if on average we wish to tolerate an error of e in each dimension, we should set $\varepsilon \simeq e\sqrt{n_\mathcal{X}}$. In our test cases we have used $\varepsilon = 0.05\,\sqrt{n_\mathcal{X}}$. Finally, ρ only plays a relevant role on almost flat manifolds. The only restriction to consider is that ρ must be smaller than σ to ensure the eventual creation of new charts. Following [12], we suggest to set $\rho = d_\mathcal{X}/2$. With this value, charts are generally created before the numerical process implementing Eq. (9)

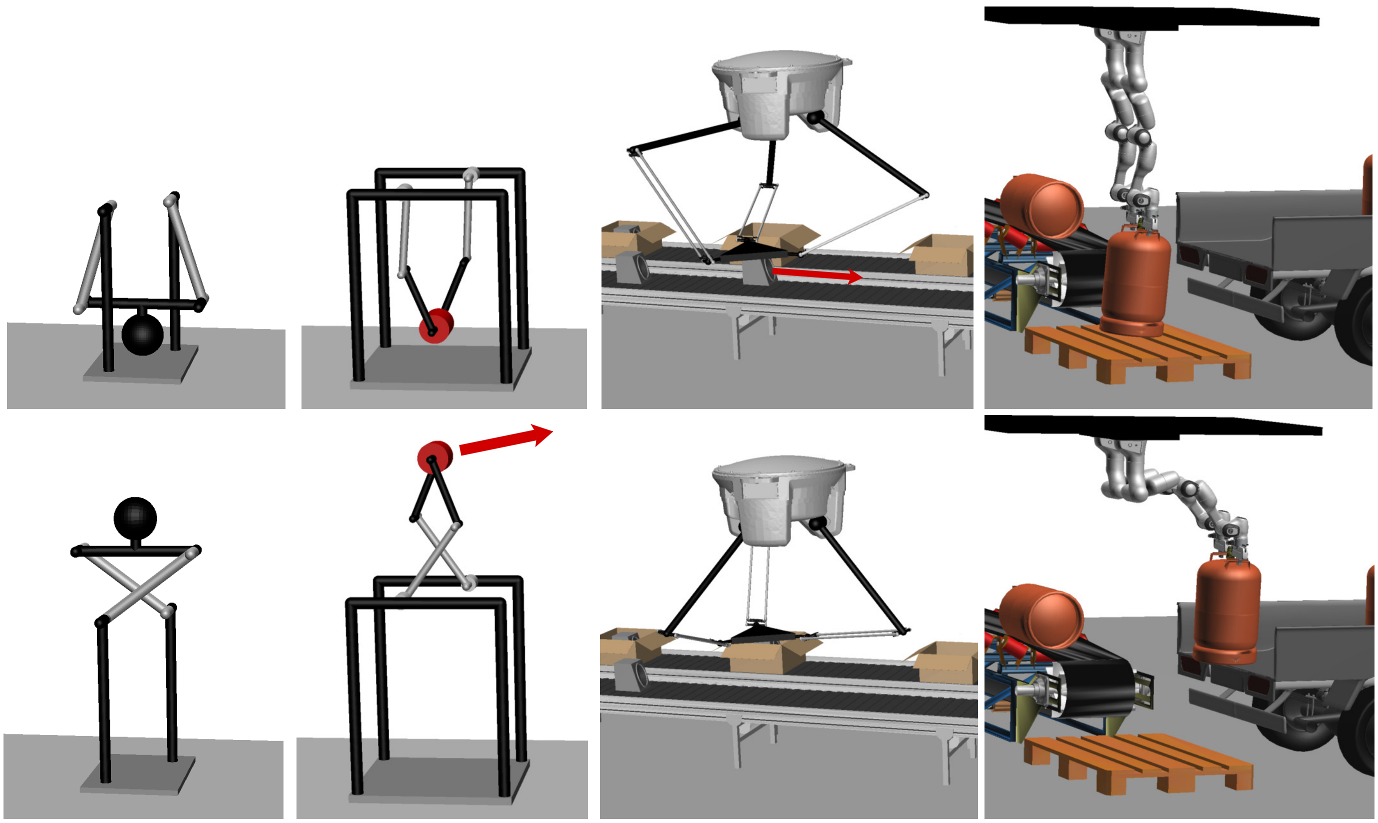


Fig. 10. Example tasks used to illustrate the performance of the planner. From left to right, and columnwise: weight lifting, weight throwing, conveyor switching, and truck loading. The robots involved are, respectively, a four-bar robot, a five-bar robot, a Delta robot, and a double-arm manipulation 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.

fails and before Eqs. (11) and (12) hold. In this way, the paving of the manifold tends to be more regular.

As explained in Section VII-A, the sampling radius σ used in line 3 of Algorithm 2 controls the exploration bias of the algorithm. The role of σ is equivalent to that of the parameter used in standard RRTs to limit the sampling space (e.g., the boundaries of a 2D maze where a mobile robot is set to move). A too large σ complicates the solution of problems with narrow corridors. Thus, we propose to set $\sigma = 2\rho$ since this a moderate value that still creates a strong push towards unexplored regions, specially in large dimensional state spaces. If necessary, existing techniques to automatically tune this parameter [59] could be adapted to kinodynamic planning.

Parameter δ appears in line 12 of Algorithm 3 and in line 4 of Algorithm 4. An equivalent parameter is present in the standard RRT algorithm [7]. If two states are closer than δ, they are considered to be close enough so that the transition between them is not problematic. Thus, this parameter is used as an upper bound of the distance between consecutive states along an RRT branch. Therefore, the value of h in Eq. (36) is adjusted so that $\|\boldsymbol{\varphi}_c(\boldsymbol{x}_k) - \boldsymbol{\varphi}_c(\boldsymbol{x}_{k+1})\| < \delta$. Moreover, to correctly detect the transition between charts, δ must be significantly smaller than ρ. With these considerations in mind, we propose to set $\delta \simeq 0.02\,\rho$.

Parameter β appears in line 11 of Algorithm 1 and is the tolerated error in the connection between trees. This parameter is also used in the standard RRT algorithm. A small value may unnecessarily complicate some problems, specially if the steering algorithm is not very precise (like in randomised steering), and a large value may produce unfeasible solutions. We suggest to use $\beta = 0.1\,\sqrt{n_{\mathcal{X}}}$, but this value has to be tunned according to the particularities of the obstacles in the environment.

Matrix $\boldsymbol{R}$ in Eq. (17) is used in the standard LQR to penalise the control effort employed and is typically initialised using the Bryson rule [60]. Finally, t_{max} fixes the time window over which $J(t_f)$ in Eq. (33) is to be minimised. Ideally, it should be slightly larger than t_f^*. A much larger value would result in a waste of computational resources and a too low value would produce sub-optimal polices. We propose to set this parameter to a fraction of the expected trajectory time t_g.

VIII. Probabilistic Completeness

In its fully randomised version, i.e., when using randomised steering instead of LQR steering, the planner is probabilistically complete. A formal proof of this point would replicate the same arguments used in [61] with minor adaptations, so we only sketch the main points supporting the claim.

Assume that the action to execute is selected at random from $\mathcal{U}$, with a random time horizon. Then, in the part

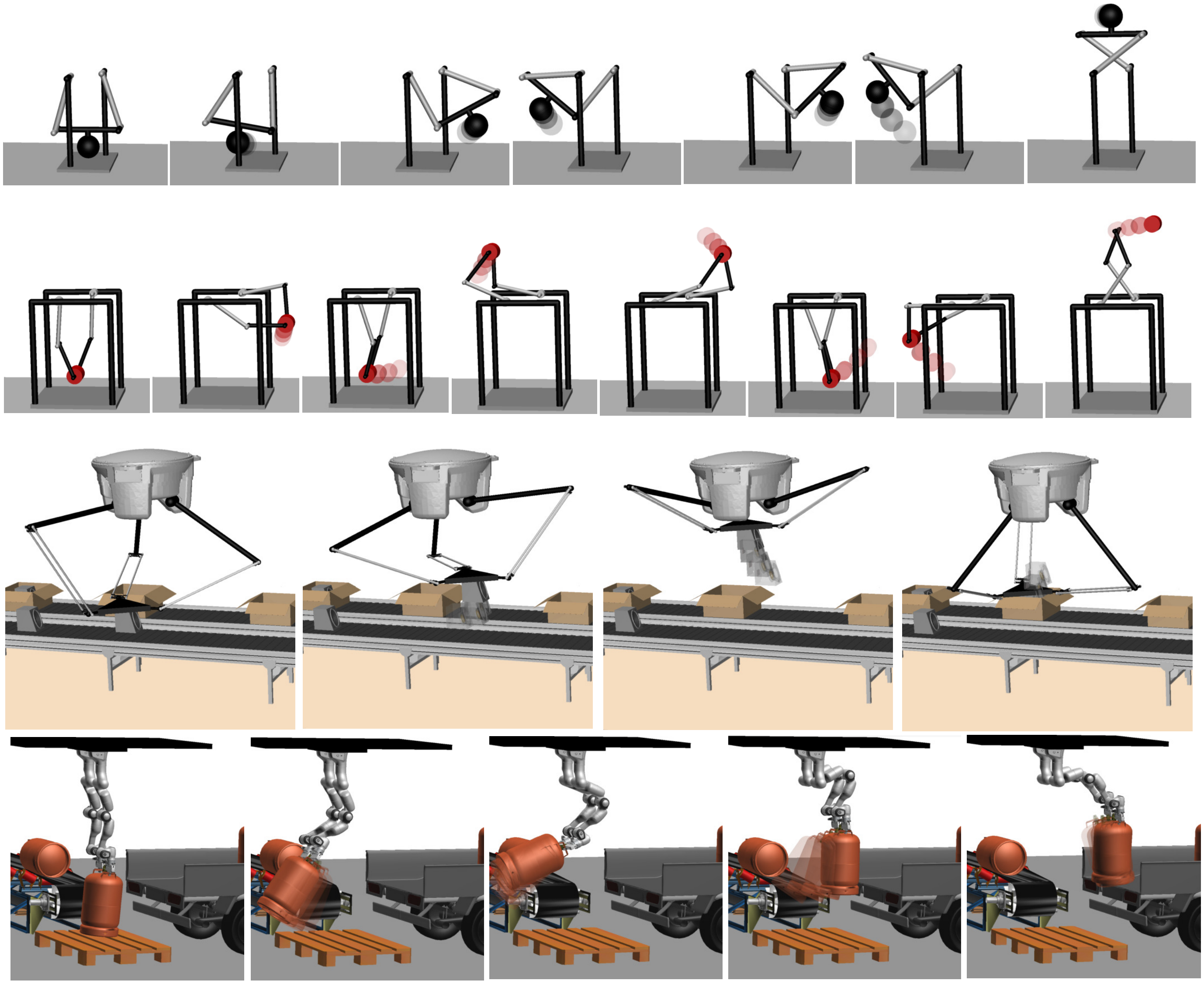


Fig. 11. Solution trajectories for the four test cases. The shown trails depict earlier positions of the load during a same time span. A longer trail, therefore, corresponds to a higher velocity of the load. See https://youtu.be/-_DMzK5SGzQ for an animated version of this figure.

of $\mathcal{X}$ already covered by a partial atlas, we are in the same situation as the one considered in [61, Section IV]: $\mathcal{X}$ is a smooth manifold, we have a procedure to sample $\mathcal{X}$, Euclidean distance is used to determine nearest neighbours, and the motion of the system is governed by a differential equation depending on the state and the control inputs. The main relevant difference is that our sample distribution is uniform in tangent space, but not on $\mathcal{X}$. However, the difference between the uniform distribution in parameter space and the actual distribution on the manifold is bounded by parameter α [62]. Thus, the probability bounds given in [61] may need to be modified, but their proof would still hold. Thus, under the same mild conditions assumed in [61] (i.e., Lipschitz-continuity conditions), our planner with randomised steering is probabilistically complete in, at least, the part of the manifold already covered by the atlas. This implies that the planner will be probabilistically complete provided it is able to extend the atlas to cover $\mathcal{X}$ completely. But this will certainly be achieved in the limit, as the procedure described in Section V-A ensures that new charts are generated each time the RRT branches approach the border of the subset of $\mathcal{X}$ covered up to a given moment. The reasoning in [61] can also be used to provide a formal proof that the tree will eventually reach such border regions, just by defining goal areas in them. Moreover, as shown in [53], the expansion of the atlas will stop when the atlas has no border, i.e., when it fully covers $\mathcal{X}$. The chart coordination procedure described in Section V-C may leave uncovered areas on the manifold, of size $O(\alpha)$ [53]. However, such areas can be detected during tree extension, and can be eliminated by slightly enlarging the P_c sets of the charts around them, as described in [12].

In principle, the use of LQR steering instead of randomised steering can only result in better performance, as it should facilitate the connection between the balls used in [61, The-

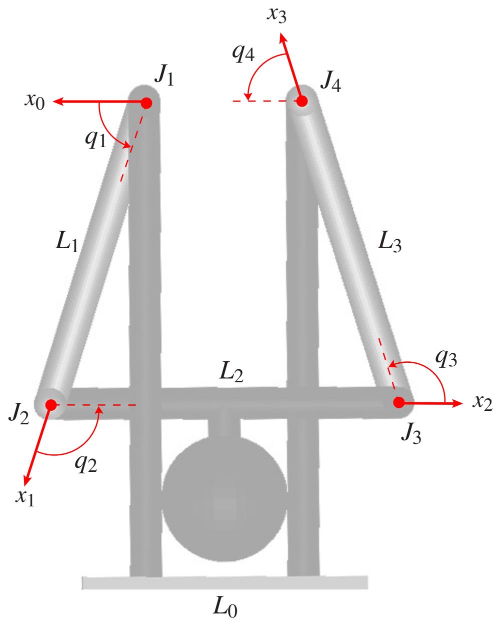


Fig. 12. Geometry of the four-bar mechanism in Fig. 10, left column. For each coordinate system, only the x axis is depicted.

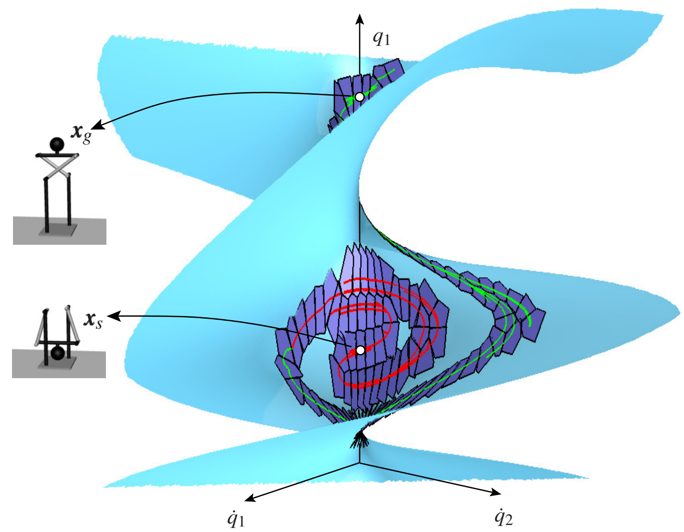


Fig. 13. A partial atlas of $\mathcal{X}$ used to plan the lifting of a weight with the four-bar robot. The red and green trees are rooted at $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$ respectively, and they are grown towards each other in parallel with the atlas. Each polygon in dark blue corresponds to the P_c set of a given chart. To allow a clearer visualization of the atlas, we have used $\rho = 0.5$ to obtain the plot, so the shown charts are actually smaller than those used by the planner. See https://youtu.be/-_DMzK5SGzQ for an animated version of this figure.

orem 2] to cover the solution trajectory: connecting them using LQR steering should be easier than doing so with randomised steering. However, a formal proof of this point would require to provide error bounds for the LQR steering policies analogous to those in [61, Lemma 3] for randomly-selected constant actions. As in [61], the obtention of such bounds remains as an open problem however, so we only conjecture the planner to be probabilistically complete if LQR steering is used. Even so, note we could always retain the probabilistic completeness by using randomised steering once in a while, instead of using LQR steering exclusively.

IX. Planning Examples

The planner has been implemented in C and it has been integrated into the CUIK suite [63]. We next analyse its performance in planning four tasks of increasing complexity (Fig. 10). The first two tasks involve planar single-loop mechanisms, which are simple enough to illustrate key aspects of the planner, like the performance of the steering method, the traversal of singularities, or the ability to plan trajectories towards states of nonzero velocity. The third and fourth tasks, on the other hand, show the planner performance in spatial robots of considerable complexity. In all cases 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. Following Section VII-C, we have fixed $\cos(\alpha) = 0.9$, $\varepsilon = 0.05\sqrt{n_{\mathcal{X}}}$, $\rho = d_{\mathcal{X}}/2$, $\sigma = 2\rho$, $\delta = 0.02\rho$, $\beta = 0.1\sqrt{n_{\mathcal{X}}}$. Matrix $\boldsymbol{R}$ in Eq. (17) has been set to be diagonal, with $\boldsymbol{R}_{i,i} = 1/{l_i}^2$, and we use $t_{max} = 1.5$. The planner performance, however, does not depend on these parameters exclusively, but also on the peculiarities of each problem, like the torque limits of the actuators, the system masses, or the presence of obstacles. An example of the trajectories we obtain can be seen in Fig. 11 and in the companion video of this paper (also available in https://youtu.be/-_DMzK5SGzQ). The complete set of geometric and dynamic parameters of all examples is provided in http://www.iri.upc.edu/cuik.

Table I summarises the problem dimensions and performance statistics for the four mentioned tasks. For each task we provide the number of generalised 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 specific formulations used for Eqs. (1) and (2) are given in [48]. 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. The column "Success rate" gives the percentage of planner runs that were able to solve each problem in at most one hour. Statistics for both the randomised steering strategy in [18] and the LQR steering strategy of this paper are given for comparison. The randomised strategy employs $2\,n_u$ random actions from $\mathcal{U}$, which are applied during 0.1 seconds in accordance with [18]. As seen in the table, the LQR strategy is more efficient than the randomised strategy, as it requires a smaller number of samples and charts, and less time, to find a solution. In fact, the success rate of the randomised strategy is only 55% in the truck loading task. Further details on the four tasks are next provided.

A. Weight lifting

The first task to be planned consists in lifting a heavy load with a four-bar robot (Fig. 10, left column). The robot involves four links cyclically connected with revolute joints from which only joint J_1 is actuated (Fig. 12). The relative angle with the

TABLE I
PROBLEM DIMENSIONS AND PERFORMANCE STATISTICS FOR THE EXAMPLE TASKS.

					Randomised 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	525	94	0.89	100%	66	46	0.17
Weight throwing	5	3	4	2	17085	5358	359.95	100%	1250	607	9.83
Conveyor switching	15	12	6	3	24247	256	420.13	100%	1809	165	40.89
Truck loading	10	6	8	10	11600	1282	1649.86	55%	9779	1269	105.47

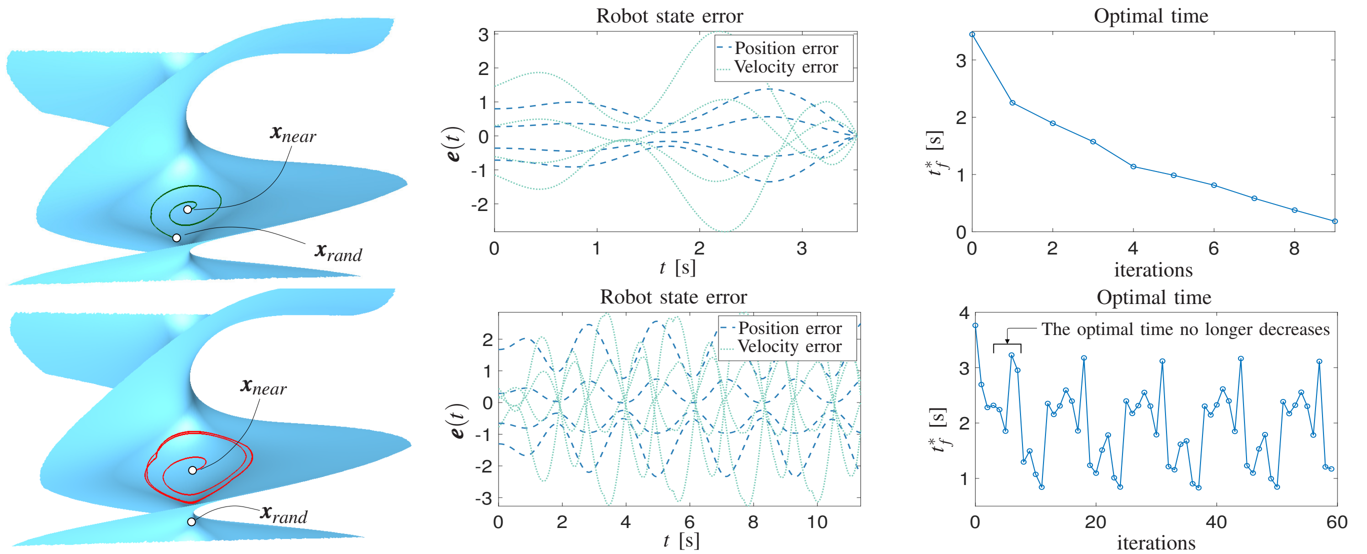


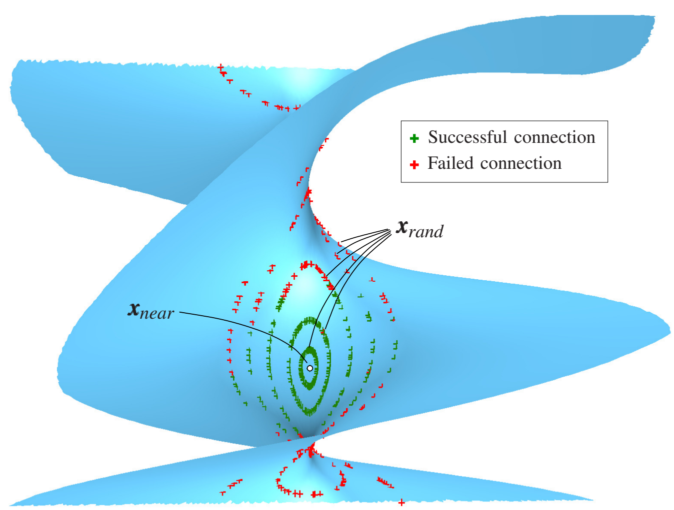
Fig. 14. Steering the four-bar robot from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{rand}$. Top: The LQR strategy allows the planner to connect $\boldsymbol{x}_{near}$ and $\boldsymbol{x}_{rand}$. Bottom: The strategy enters a limit cycle and is never able to reach $\boldsymbol{x}_{rand}$. The right plot shows that t_f^* no longer decreases after six iterations, so it would be aborted at this point.

following link is denoted by q_i, and the robot configuration is then given by $\boldsymbol{q} = (q_1, q_2, q_3, q_4)$.

Figure 13 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. A typical trajectory returned by the planner can be seen in Fig. 11, top row.

This example can be used to illustrate the performance of the LQR steering strategy. Fig. 14-top, shows an example in which this strategy successfully finds a trajectory connecting $\boldsymbol{x}_{near}$ with $\boldsymbol{x}_{rand}$, i.e., the error $\boldsymbol{e}(t) = \boldsymbol{x}(t) - \boldsymbol{x}_{rand}$ converges to $\mathbf{0}$. Whenever a new chart is created during the simulation or when a chart is revisited, the policy is recomputed and, in this example, t_f^* monotonically decreases. In contrast, Fig. 14-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, i.e., the error $\boldsymbol{e}(t)$ is never $\mathbf{0}$. The steering method in Algorithm 3 would stop as soon as t_f^* no longer decreases.

In Fig. 15 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



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

Fig. 15. Success rate of the LQR steering strategy for states $\boldsymbol{x}_{rand}$ that are increasingly far from $\boldsymbol{x}_{near}$.

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.

B. 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. 10, 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 computed trajectory can be seen in the second row of Fig. 11. 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 specific velocities and accelerations fulfilling certain geometric conditions [17, 64]. 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 [65]. In the trajectory shown in Fig. 11 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 [66], recent LQR controllers for closed kinematic chains have no trouble in accomplishing this task [67].

C. 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. 10, third column).

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. Moreover, the boxes have thin walls that require us to set $\beta = 0.05\sqrt{n_{\mathcal{X}}}$ to avoid obtaining unfeasible solutions. An example of a resulting trajectory can be seen in Fig. 11, 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.

D. 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 randomised 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, [7] proposes to simulate only $2\,n_u$ random actions for each branch extension. Our implementation adopts this criterion but, like [7], 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 I). 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 I, last column). Using this strategy, the planner obtained trajectories like the one shown in Fig. 11, 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.

X. Conclusions

This paper 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 to 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 and the force bounds of the actuators. In its fully randomised version (i.e., using randomised steering), the planner is probabilistically complete. We also conjecture it is probabilistically complete if LQR steering is used, but proving this point remains open so far. The examples in the paper 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 optimised in some sense, minimising the time or control effort required for its execution. Trajectory optimization tools like those in [38], [41], or [42] might be very helpful to both ends. Another sensitive point is the metric employed to measure the distance between states. This is a general concern in any motion planner, but it is more difficult to address in our context as such metric should not only 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 monitoring of constraint forces during the extension of the RRT. While such forces result in no motion, they do stress the robot parts unnecessarily and should be kept under admissible bounds. Note that these forces can be computed as the simulations proceed, since they can be inferred, e.g., from the values of the Lagrange multipliers involved in the equations of motion [68]. The ability to impose bounds on constraint forces would also allow computing trajectories in closed kinematic chains induced by unilateral contacts, like those that arise when a hand moves an object in contact with a surface. Such contacts could be maintained along a trajectory by setting pertinent signed bounds on the constraint forces arising.

References

[1] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM*, vol. 40, no. 5, pp. 1048–1066, 1993.

[2] S.-H. Lee, J. Kim, F. C. Park, M. Kim, and J. E. Bobrow, "Newton-type algorithms for dynamics-based robot movement optimization," *IEEE Transactions on robotics*, vol. 21, no. 4, pp. 657–667, 2005.

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

[4] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, "Optimization based full body control for the atlas robot," in *IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 120–127.

[5] M. A. Diftler, J. S. Mehling, M. E. Abdallah, N. A. Radford, L. B. Bridgwater, A. M. Sanders, R. S. Askew, D. M. Linn, J. D. Yamokoski, F. A. Permenter, B. K. Hargrave, R. Platt, R. T. Savely, and R. O. Ambrose, "Robonaut 2 - the first humanoid robot in space," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2178–2183.

[6] R. P. Hoyt, "Spiderfab: An architecture for self-fabricating space systems," in *AIAA Space 2013 Conference and Exposition*, 2013, p. 5509.

[7] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[8] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[9] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1431–1443, 2016.

[10] D. Berenson, S. Srinivasa, and J. J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[11] J. M. Porta, L. Jaillet, and O. Bohigas, "Randomized path planning on manifolds based on higher-dimensional continuation," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 201–215, 2012.

[12] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 105–117, 2013.

[13] K. Hauser, "Fast interpolation and time-optimization with contact," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1231–1250, 2014.

[14] Q.-C. Pham and O. Stasse, "Time-optimal path parameterization for redundantly actuated robots: A numerical integration approach," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 6, pp. 3257–3263, 2015.

[15] Z. Kingston, M. Moll, and L. E. Kavraki, "Decoupling constraints from sampling-based planners," in *International Symposium of Robotics Research*, 2017.

[16] ——, "Sampling-based methods for motion planning with constraints," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 159–185, 2018.

[17] O. Bohigas, M. Manubens, and L. Ros, *Singularities of robot mechanisms: Numerical computation and avoidance path planning*, ser. Mechanisms and Machine Science. Springer, 2017, vol. 41.

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

[19] J. H. Yakey, S. M. LaValle, and L. E. Kavraki, "Randomized path planning for linkages with closed kinematic chains," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 951–959, 2001.

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

[21] 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*, 2002, pp. 2141–2146.

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

[23] B. Kim, T. T. Um, C. Suh, and F. C. Park, "Tangent bundle RRT: A randomized algorithm for constrained motion planning," *Robotica*, vol. 34, no. 1, pp. 202–225, 2016.

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

[25] 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*, ser. Intelligent Robotics and Autonomous Agents. MIT Press, 2005.

[26] J. E. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.

[27] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Transactions on Automatic Control*, vol. 30, no. 6, pp. 531–541, 1985.

[28] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal on Robotics and Automation*, vol. 3, no. 2, pp. 115–123, 1987.

[29] J.-J. Slotine and H. S. Yang, "Improving the efficiency of time-optimal path-following algorithms," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 118–124, 1989.

[30] Z. Shiller and H.-H. Lu, "Computation of path constrained time optimal motions with dynamic singularities," *Journal of Dynamic Systems, Measurement, and Control*, vol. 114, no. 1, pp. 34–40, 1992.

[31] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, 2014.

[32] Q.-C. Pham, S. Caron, P. Lertkultanon, and Y. Nakamura, "Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots," *The International Journal of Robotics Research*, vol. 36, no. 1, pp. 44–67, 2017.

[33] J. E. Bobrow, "Optimal robot path planning using the minimum-time criterion," *IEEE Journal on Robotics and Automation*, vol. 4, no. 4, pp. 443–450, 1988.

[34] Z. Shiller and S. Dubowsky, "Robot path planning with obstacles, actuator, gripper, and payload constraints," *The International Journal of Robotics Research*, vol. 8, no. 6, pp. 3–18, 1989.

[35] 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*, vol. 7, no. 6, pp. 785–797, Dec 1991.

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

[37] 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*, 2011, pp. 4569–4574.

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

[39] J. Schulman, D. Y, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[40] 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*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

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

[42] ——, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[43] 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*, 2014, pp. 3713–3719.

[44] 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*, 2012, pp. 2537–2542.

[45] R. Tedrake, "LQR-Trees: Feedback motion planning on sparse randomized trees," in *Robotics: Science and Systems*, 2009.

[46] 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*, 2013, pp. 2429–2436.

[47] 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*, 2013, pp. 5054–5061.

[48] R. Bordalba, L. Ros, and J. M. Porta, "A randomized kinodynamic planner for closed-chain robotic systems," Institut de Robòtica i Infomàtica. CSIC-UPC. Technical Report, 2019, https://bit.ly/2vk4MSO.

[49] F. A. Potra and J. Yen, "Implicit numerical integration for Euler-Lagrange equations via tangent space parametrization," *Journal of Structural Mechanics*, vol. 19, no. 1, pp. 77–98, 1991.

[50] L. R. Petzold, "Numerical solution of differential-algebraic equations in mechanical systems simulation," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1–4, pp. 269–279, 1992.

[51] R. Featherstone and D. E. Orin, "Dynamics," in *Springer Handbook of Robotics*. Springer, 2016, pp. 37–66.

[52] W. Blajer, "Methods for constraint violation suppression in the numerical simulation of constrained multibody systems - A comparative study," *Computer Methods in Applied Mechanics and Engineering*, vol. 200, no. 13-16, pp. 1568–1576, 2011.

[53] M. E. Henderson, "Multiple parameter continuation: computing implicitly defined k-manifolds," *International Journal of Bifurcation and Chaos*, vol. 12, no. 3, pp. 451–476, 2002.

[54] E. Hairer, "Geometric integration of ordinary differential equations on manifolds," *BIT Numerical Mathematics*, vol. 41, no. 5, pp. 996–1007, 2001.

[55] E. Hairer, C. Lubich, and G. Wanner, "Geometric numerical integration: structure-preserving algorithms for ordinary differential equations," 2006.

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

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

[58] J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, vol. 2, 2000, pp. 995–1001.

[59] J. M. Porta and L. Jaillet, "Sampling strategies for path planning under kinematic constraints," *CoRR*, vol. abs/1407.2544, 2014.

[60] A. E. Bryson and Y.-C. Ho, *Applied Optimal Control: Optimization, Estimation and Control*. Taylor Francis, 1975.

[61] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin, "Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 277–283, 2019.

[62] L. Jaillet and J. M. Porta, "Efficient asymptotically-optimal path planning on manifolds," *Robotics and Autonomous Systems*, vol. 61, no. 8, pp. 797–807, 2013.

[63] J. M. Porta, L. Ros, O. Bohigas, M. Manubens, C. Rosales, and L. Jaillet, "The Cuik Suite: Analyzing the motion of closed-chain multibody systems," *IEEE Robotics and Automation Magazine*, vol. 21, no. 3, pp. 105–114, 2014.

[64] S. Briot and V. Arakelian, "Optimal force generation in parallel manipulators for passing through the singular positions," *The International Journal of Robotics Research*, vol. 27, no. 8, pp. 967–983, 2008.

[65] F. Bourbonnais, P. Bigras, and I. A. Bonev, "Minimum-time trajectory planning and control of a pick-and-place five-bar parallel robot," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 2, pp. 740–749, 2015.

[66] F. Aghili, "A unified approach for inverse and direct dynamics of constrained multibody systems based on linear projection operator: applications to control and simulation," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 834–849, 2005.

[67] 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*, 2018, pp. 270–276.

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