Randomized Kinodynamic Planning for Constrained Systems

Ricard Bordalba, Lluís Ros, and Josep M. Porta

Abstract— **Kinodynamic RRT planners are considered to be general tools for effectively finding feasible trajectories for high-dimensional dynamical systems. However, they struggle when holonomic constraints are present in the system, such as those arising in parallel manipulators, in robots that cooperate to fulfill a given task, or in situations involving contacts with the environment. In such cases, the state space becomes an implicitly-defined manifold, which makes the diffusion heuristic inefficient and leads to inaccurate dynamical simulations. To address these issues, this paper presents an extension of the kinodynamic RRT planner that constructs an atlas of the state-space manifold incrementally, and uses this atlas both to generate random states and to dynamically steer the system towards such states. To the best of our knowledge, this is the first randomized kinodynamic planner that explicitly takes holonomic constraints into account. We validate the approach in significantly-complex systems.**

I. Introduction

The motion planning problem has been a subject of active research since the early days of robotics [1]. Although it can be formulated in simple terms—find a feasible trajectory to move a robot between two states—and despite the significant advances in the field, it is still an open problem in many respects. The complexity of the problem arises from the multiple constraints that have to be taken into account, like potential collisions with static or moving objects in the environment, loop-closure constraints, dynamic equations, torque and velocity limits, or energy and time execution bounds, to name a few. Often, such a complexity is managed by relaxing some of the constraints. For example, while obstacle avoidance is a fundamental issue, lazy approaches initially disregard it [2]. Other approaches concentrate on kinematic feasibility [3], which is already a challenging issue by itself. In these and other approaches, dynamic constraints such as speed, acceleration, or torque limits are neglected, with the hope that they will be enforced in a post-processing stage, using dynamic time-scaling methods for example [4]. Decoupled approaches, however, may not lead to solutions satisfying all the constraints. It is not difficult to find situations in which a kinematically-feasible path cannot be transformed into a time-parametric trajectory compatible with the system dynamics. For this reason, substantial efforts have also been devoted to obtaining so-called kinodynamic planners, which directly synthesize state trajectories simultaneously compatible with as many kinematic and dynamic constraints as possible [5], [6], [7], [8].

Among all kinodynamic planning approaches, the rapidly-exploring random tree (RRT) method [6] has emerged as one of the most successful algorithms. Kinodynamic RRT planners are conceptually simple, easy to implement, and effective, even in high dimensions. Often, such planners are thought to be quite general, being able to accommodate most of the motion planning constraints needed in practice. While it is true that kinodynamic RRTs cover many situations, they suffer from an important limitation: they assume that the state space can be described parametrically, or, in other words, that the robot state can be represented by means of *independent* generalized coordinates. Although parametric state spaces arise frequently, for example in single-body robots, or in articulated robots with tree topology, holonomic constraints may also appear that relate the state space coordinates in nontrivial ways. This occurs, for example, in systems with closed kinematic chains, in robots in contact with the environment, or when geometric constraints are needed to fulfill a given task. In these cases, the robotic system is said to be *constrained*, because its state space is a manifold implicitly-defined by a system of nonlinear equations.

Standard kinodynamic RRT methods are in trouble on constrained systems: their diffusion heuristic becomes inefficient, they may fail to find feasible motions when they exist, and easily produce simulations that violate the holonomic constraints. A goal of this paper is to show that these difficulties can all be circumvented if the differential geometric structure of the state space is considered inside the planner. While some approaches treat holonomic constraints [9], [10], [11], [12], [13], [14], none of them considers the dynamics of the system into the planner. This paper extends the methods in [14] to obtain a randomized kinodynamic planner that simultaneously enforces holonomic and dynamic constraints. This planner can also be seen as an extension of the one in [6] to deal with holonomic constraints.

The rest of the paper is organized as follows. Section II formulates the motion planning problem on constrained robotic systems with dynamic constraints. Then, Section III illustrates the mentioned difficulties of the classic kinodynamic RRT method [6] when applied to such systems. Sections IV and V show how this method can be duly extended to overcome these difficulties. The idea is to construct an atlas of the state space incrementally, and then use this atlas to efficiently push the growing of the tree towards unexplored regions, while at the same time performing accurate dynamic simulations. The planner is validated on a number of test cases in Section VI, and Section VII finally concludes the paper, discussing points for further attention.

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness under projects DPI2014-57220-C2-2-P, DPI2017-88282-P, and MDM-2016-0656.

Ricard Bordalba, Lluís Ros, and Josep M. Porta are with the Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Barcelona, Spain {rbordalba,porta,ros}@iri.upc.edu

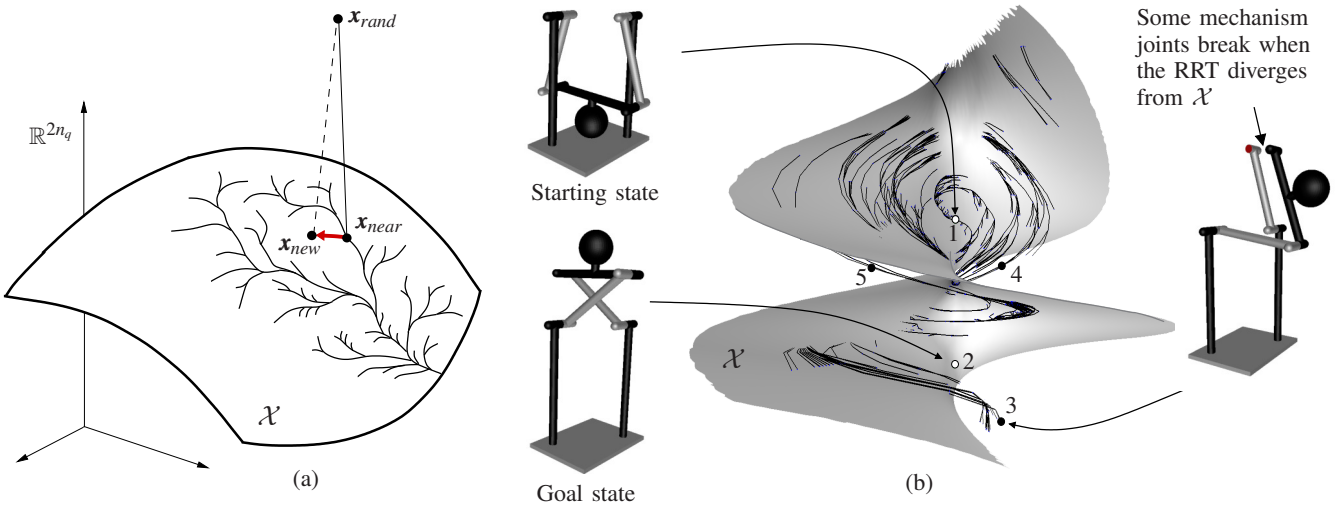

Fig. 1. Drawbacks of the standard RRT method when applied to constrained systems. See the text for details. Note that, on the right figure, the RRT easily diverges from $\mathcal{X}$, as revealed by the fact that it often gets hidden beneath the gray surface.

II. Problem formulation

Let us describe a robot configuration by means of a tuple $\boldsymbol{q}$ of n_q generalized coordinates, which determine the positions and orientations of all links at a given instant of time. We restrict our attention to constrained systems, i.e., those in which $\boldsymbol{q}$ must satisfy a system of n_e nonlinear equations

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

encompassing all holonomic constraints to be taken into account, either inherent to the robot design (like closed kinematic chains) or necessary for task execution (like geometric or contact constraints imposed on the end-effector). Then, the configuration space $\mathcal{C}$ of the robot, or C-space for short, is the nonlinear variety $\mathcal{C} = \{\boldsymbol{q} : \boldsymbol{\Phi}(\boldsymbol{q}) = \boldsymbol{0}\}$, which may be quite complex in general. Under mild conditions, however, we can assume that the Jacobian $\boldsymbol{\Phi_q}(\boldsymbol{q}) = \partial\boldsymbol{\Phi}/\partial\boldsymbol{q}$ is full rank for all $\boldsymbol{q} \in \mathcal{C}$, so that $\mathcal{C}$ is a smooth manifold of dimension $d_{\mathcal{C}} = n_q - n_e$. By differentiating Eq. (1) with respect to time we obtain

$$\boldsymbol{\Phi_q}(\boldsymbol{q})\, \dot{\boldsymbol{q}} = \boldsymbol{0}, \tag{2}$$

which delimits the feasible velocity vectors $\dot{\boldsymbol{q}}$ at a given $\boldsymbol{q} \in \mathcal{C}$. Now, let $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$ denote the system formed by Eqs. (1) and (2), where $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{2n_q}$. While [14] operates in $\mathcal{C}$, our planning problem will take place in the state space

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

Since $\boldsymbol{\Phi_q}(\boldsymbol{q})$ is full rank, $\mathcal{X}$ is also a smooth manifold of dimension $d_{\mathcal{X}} = 2\, d_{\mathcal{C}}$, which implies that the tangent space of $\mathcal{X}$ at $\boldsymbol{x}$,

$$\mathcal{T}_{\boldsymbol{x}}\mathcal{X} = \{\dot{\boldsymbol{x}} \in \mathbb{R}^{2n_q} : \boldsymbol{F_x}(\boldsymbol{x})\, \dot{\boldsymbol{x}} = \boldsymbol{0}\}, \tag{4}$$

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

We shall encode the forces and torques of the actuators into an action vector $\boldsymbol{u}$ of dimension 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 constrained system is determined by a differential-algebraic equation of the form

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

Eq. (5) forces the states $\boldsymbol{x}$ to remain in $\mathcal{X}$, while Eq. (6) models the dynamics of the system, and can be obtained from the multiplier form of the Euler-Lagrange equations [15]. For each value of $\boldsymbol{u}$, Eq. (6) defines a vector field over $\mathcal{X}$, which can be used together with Eq. (5) to integrate the robot motion forward in time, using proper numerical methods.

To model the fact that the actuator forces are limited in practice, we assume that $\boldsymbol{u}$ takes values in some bounded subset $\mathcal{U}$ of $\mathbb{R}^{n_u}$, which indirectly limits the acceleration of the system. During its motion, moreover, the robot cannot incur in collisions with itself or with the environment, which reduces the feasible states $\boldsymbol{x}$ to those lying in a subset $\mathcal{X}_{\text{free}} \subseteq \mathcal{X}$ of non-collision states, which should always fulfill any existing limits on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$.

With the previous definitions, the planning problem we confront can be phrased as follows. Given two states of $\mathcal{X}_{\text{free}}$, $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$, find an action trajectory $\boldsymbol{u} = \boldsymbol{u}(t) \in \mathcal{U}$ such that the trajectory $\boldsymbol{x} = \boldsymbol{x}(t)$ determined by Eqs. (5) and (6) for $\boldsymbol{x}(0) = \boldsymbol{x}_s$ fulfills $\boldsymbol{x}(t_f) = \boldsymbol{x}_g$ for some time $t_f > 0$, and $\boldsymbol{x}(t) \in \mathcal{X}_{\text{free}}$ for all $t \in [0, t_f]$.

III. Drawbacks of the standard RRT method

Observe from the previous section that, in contrast to [6], we allow the presence of Eq. (1) in the formulation of our planning problem, which makes it more general and challenging at the same time. In the literature, the suggested

way to handle this equation is to differentiate it twice, and use it in conjunction with the Euler-Lagrange equations with multipliers to obtain the explicit form of the motion equation [16, Sec. 13.4.3.1.]. In fact, this is the process that we follow to obtain Eq. (6). However, the application of the standard RRT method to this equation alone, disregarding Eq. (5), presents the following drawbacks.

On the one hand, the random samples used to guide the RRT extension would not be generated on $\mathcal{X}$, but in the larger ambient space $\mathbb{R}^{2n_q}$, which results in an inefficient exploration of $\mathcal{X}$ [14], [17]. This can be seen in Fig. 1(a), in which a partial RRT has been grown on $\mathcal{X}$. Clearly, there is a high probability of producing an ambient space sample $\boldsymbol{x}_{rand}$ such that the nearest RRT node, $\boldsymbol{x}_{near}$, will only be expanded slightly towards a new node $\boldsymbol{x}_{new}$.

On the other hand, note that the standard RRT method would only use Eq. (6) to simulate the motion of the system, treating it as an ordinary differential equation. However, from multibody mechanics it is known that the motion of a constrained system can only be predicted reliably if Eq. (5) is also taken into account during the integration of Eq. (6) [18]. Otherwise, the inevitable errors introduced when discretizing Eq. (6) will make the trajectory $\boldsymbol{x}(t)$ increasingly drift away from $\mathcal{X}$ as the simulation progresses. This phenomenon is shown in Fig. 1(b) for a four-bar pendulum modeling a swing-boat ride. The pendulum has to be moved from the starting to the goal states indicated (points 1 and 2 of $\mathcal{X}$), both with zero velocity. As shown, an RRT built by the method in [6] easily diverges from $\mathcal{X}$ as the planner proceeds (e.g., around points 3, 4, and 5) and, as a result, the query states cannot be connected reliably.

IV. Mapping and Exploring the State Space

We will next see that the sampling and drift issues just mentioned can both be circumvented by constructing an atlas of $\mathcal{X}$. The atlas will provide us with a means to sample the $\mathcal{X}$ manifold, instead of the larger ambient space. In addition, the atlas charts will permit the integration of Eqs. (5) and (6) as a true differential-algebraic equation, guaranteeing a driftless simulation of the robot motions along the tree branches. These ideas will then be used in Section V to implement an RRT planner for constrained systems.

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. 2(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 $\boldsymbol{\psi}_c = \boldsymbol{\varphi}_c^{-1}$ exists and gives a local parameterization of V_c.

To construct $\boldsymbol{\varphi}_c$ and $\boldsymbol{\psi}_c$ we shall use the so-called tangent space parameterization [15], [19]. In this approach, 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. 2(b)]. Thus $\boldsymbol{\varphi}_c$ becomes

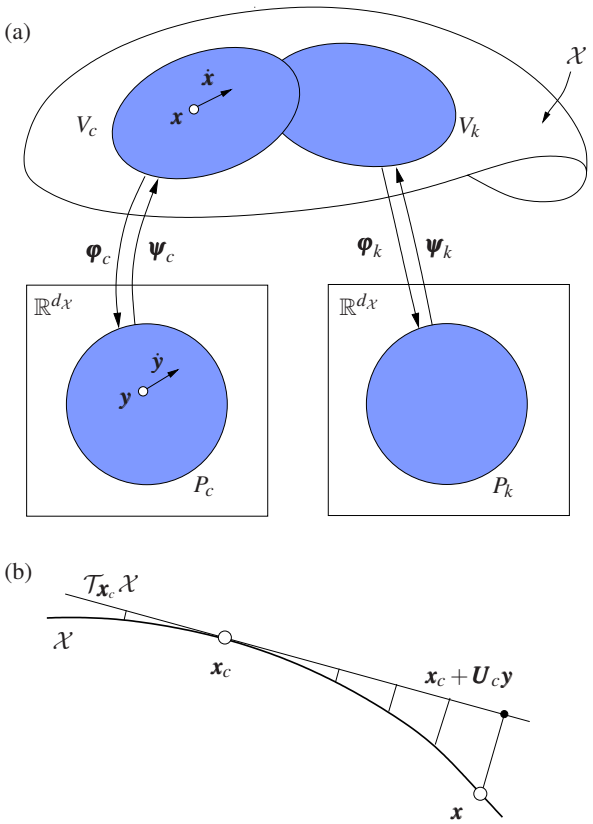

Fig. 2. (a) An atlas is a collection of maps $\boldsymbol{\varphi}$ providing local coordinates to all points of $\mathcal{X}$. (b) The projection of the points $\boldsymbol{x} \in \mathcal{X}$ to $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$ leads to specific instances of $\boldsymbol{\varphi}_c$ and $\boldsymbol{\psi}_c$.

$$\boldsymbol{y} = \boldsymbol{U}_c^\top (\boldsymbol{x} - \boldsymbol{x}_c), \tag{7}$$

where $\boldsymbol{U}_c$ is a $2n_q \times d_\mathcal{X}$ matrix whose columns provide an orthonormal basis of $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$. The map $\boldsymbol{x} = \boldsymbol{\psi}_c(\boldsymbol{y})$ is implicitly determined by the system of nonlinear equations

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{x}) &= \boldsymbol{0}, \\ \boldsymbol{U}_c^\top (\boldsymbol{x} - \boldsymbol{x}_c) - \boldsymbol{y} &= \boldsymbol{0}, \end{aligned} \tag{8}$$

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

Assuming that an atlas has been, the problem of sampling $\mathcal{X}$ boils down to sampling the P_c sets, since the $\boldsymbol{y}$ values can always be projected to $\mathcal{X}$ using the map $\boldsymbol{x} = \boldsymbol{\psi}_c(\boldsymbol{y})$. Also, the atlas allows the conversion of the vector field defined by Eq. (6) into one in the coordinate spaces P_c. The time derivative of Eq. (7), $\dot{\boldsymbol{y}} = \boldsymbol{U}_c^\top \dot{\boldsymbol{x}}$, gives the relationship between the two vector fields, and allows writing

$$\dot{\boldsymbol{y}} = \boldsymbol{U}_c^\top \, \boldsymbol{g}(\boldsymbol{\psi}_c(\boldsymbol{y}), \boldsymbol{u}), \tag{9}$$

which is Eq. (6), but expressed in local coordinates. This equation still takes the full dynamics into account, and forms the basis of the so-called tangent-space parameterization methods for the integration of differential-algebraic equations [20]. Given a state $\boldsymbol{x}_k$ and an action $\boldsymbol{u}$, $\boldsymbol{x}_{k+1}$ 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. (9), and finally getting $\boldsymbol{x}_{k+1} = \boldsymbol{\psi}_c(\boldsymbol{y}_{k+1})$. The procedure guarantees that $\boldsymbol{x}_{k+1} \in \mathcal{X}$ by construction, which makes the integration compliant with all kinematic constraints in Eq. (5).

B. Incremental atlas and RRT expansion

One could build a full atlas of the implicitly-defined state space and then use its local parameterizations to define a kinodynamic RRT. 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. Thus, a better alternative is to combine the construction of the atlas and the expansion of the RRT [14]. In this approach, a partial atlas is used to both generate random states and grow the RRT branches. Also, as described next, new charts are created as the RRT branches reach unexplored areas of the state space.

Suppose that $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ are two consecutive steps along an RRT branch, whose parameters in the chart defined at $\boldsymbol{x}_c$ are $\boldsymbol{y}_k$ and $\boldsymbol{y}_{k+1}$, respectively. Then, a new chart at $\boldsymbol{x}_k$ is generated if Eq. (8) cannot be solved for $\boldsymbol{x}_{k+1}$ using the Newton-Raphson method, or if any of the following conditions is met

$$\|\boldsymbol{x}_{k+1} - (\boldsymbol{x}_c + \boldsymbol{U}_c\, \boldsymbol{y}_{k+1})\| > \varepsilon, \tag{10}$$

$$\frac{\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\|}{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|} < cos(\alpha), \tag{11}$$

$$\|\boldsymbol{y}_{k+1}\| > \rho, \tag{12}$$

where ε, α, and ρ are user-defined parameters. The three conditions are introduced to ensure that the chart domains P_c capture the overall shape of $\mathcal{X}$ with sufficient detail. The first condition limits the maximal distance between the tangent space and the manifold $\mathcal{X}$. The second condition ensures a bounded curvature in the part of the manifold covered by a local parameterization, as well as a smooth transition between charts. Finally, the third condition is introduced to ensure 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, it is important to reduce the overlap between new charts and those already in the atlas. Otherwise, the areas of $\mathcal{X}$ covered by several charts would be oversampled. To avoid so, the set of valid parameters for each chart c, P_c, is defined as the intersection of a ball of radius σ centered at the origin of $\mathbb{R}^{d_\mathcal{X}}$ and a number of half-planes, all defined in $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$. The set P_c is progressively bounded as new neighboring charts are created around chart c. If, while growing an RRT branch using the local parameterization provided by $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$, a chart is created at a point $\boldsymbol{x}_k$ with parameter vector $\boldsymbol{y}_k$ in P_c, then the following inequality

$$\boldsymbol{y}^\top \boldsymbol{y}_k - \frac{\|\boldsymbol{y}_k\|^2}{2} \leq 0 \tag{13}$$

with $\boldsymbol{y} \in \mathbb{R}^{d_\mathcal{X}}$, is added to the definition of P_c (Fig. 3). A similar inequality is added to P_k, the chart at $\boldsymbol{x}_k$, by projecting $\boldsymbol{x}_c$ to $\mathcal{T}_{\boldsymbol{x}_k}\mathcal{X}$. The parameter σ must be larger than ρ to guarantee that the RRT branches in chart c will eventually trigger the generation of new charts, i.e., to guarantee that Eq. (12) eventually holds.

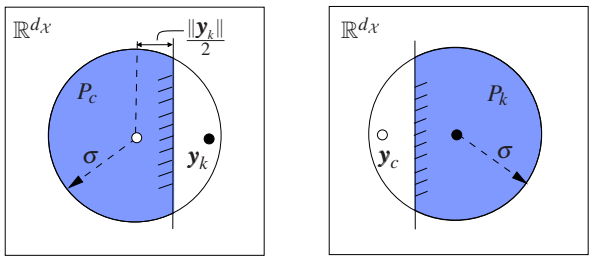

Fig. 3. Bounding of the parameter sets P_c and P_k of the two neighboring charts in Fig. 2. Note that $\boldsymbol{y}_c = \boldsymbol{\varphi}_k(\boldsymbol{x}_c)$ and $\boldsymbol{y}_k = \boldsymbol{\varphi}_c(\boldsymbol{x}_k)$.

Algorithm 1: The top-level pseudocode of the planner

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

V. Planner Implementation

Algorithm 1 gives the top-level pseudocode of the planner we propose. It can be seen that, at this level, the algorithm is almost identical to the one proposed in [6], the only difference being that we use an atlas A of $\mathcal{X}$ in our case (initialized in line 4 with one chart centered at $\boldsymbol{x}_s$ and another at $\boldsymbol{x}_g$) to support the lower-level sampling and simulation tasks. As in [6], the algorithm implements a bidirectional RRT where one tree is extended (line 8) towards a random sample (generated in line 6) and then the other tree is extended (line 10) towards the state just added to the first tree. The process is repeated until the trees become connected with a user-specified accuracy (parameter β in line 12). Otherwise, the trees are swapped (line 11) and the process is repeated. Tree extensions are always initiated at the state in the tree closer to the target state (lines 7 and 9). Different metrics can be used without affecting the overall structure of the planner. As in [6], we shall use the Euclidean distance in state space for simplicity.

A. Sampling

The atlas A is key to implement the SAMPLE method of Algorithm 1. The procedure employed is described by Algorithm 2. Initially, one of the charts covering the tree T

is selected at random with uniform distribution (line 3). A vector $\boldsymbol{y}_{rand}$ of parameters is then randomly sampled inside a ball of radius σ centered at the origin of $\mathbb{R}^{d_{\mathcal{X}}}$ (line 4), repeating this sampling if necessary until $\boldsymbol{y}_{rand}$ falls inside the set P_c for the selected chart. The procedure finally returns the point $\boldsymbol{x}_{rand} = \boldsymbol{x}_c + \boldsymbol{U}_c \boldsymbol{y}_{rand}$ corresponding to the ambient space coordinates of $\boldsymbol{y}_{rand}$ (line 7). Notice that this point lies on the tangent space $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$, instead of on $\mathcal{X}$, because the tangent space point is enough to steer the tree towards the unexplored regions.

Observe that, initially, the set P_c is the mentioned ball of radius σ centered at the origin of $\mathbb{R}^{d_{\mathcal{X}}}$. However, as new neighboring charts are successively created around a given chart (as described in Sec. IV-A), the set P_c is incrementally reduced by the addition of the limiting hyperplanes given by Eq. (13). Thus, the sets P_c of fully-surrounded charts become much smaller than the original ball of radius σ, and their probability of being sampled decreases considerably. Charts that lie at the borders of the RRT, on the contrary, have fewer neighboring charts (and thus a larger P_c set), resulting in a higher probability of being sampled. In this way, the growing of the tree is biased towards regions outside the currently-explored state space.

B. Tree extension

Algorithm 3 tries to connect a given state $\boldsymbol{x}_{near}$ with a goal state $\boldsymbol{x}_{rand}$. The procedure simulates the motion of the system (line 6) for a set of actions, which can be selected at random or taken from a predefined set (line 5). The action that yields a new state $\boldsymbol{x}_{new}$ closer to $\boldsymbol{x}_{rand}$ is added to the RRT with an edge connecting it to $\boldsymbol{x}_{near}$ (line 13). The action $\boldsymbol{u}_{new}$ generating the transition from $\boldsymbol{x}_{near}$ to the new state $\boldsymbol{x}_{new}$ is also stored in the tree so that an action trajectory can be returned after planning. This process is repeated as long as there is progress towards $\boldsymbol{x}_{rand}$.

Algorithm 4 summarizes the procedure used to simulate a given action, $\boldsymbol{u}$, from a particular state, $\boldsymbol{x}_k$. The simulation is carried out while the path is not blocked by an obstacle or by a workspace limit (line 8), while the goal state is not reached (with accuracy δ), or for a maximum time span, t_m (line 5). At each simulation step, the key procedure is the NEXTSTATE method (line 7), which provides the next state, $\boldsymbol{x}_{k+1}$, given the current one, $\boldsymbol{x}_k$, and the action to simulate, $\boldsymbol{u}$. This is implemented by integrating Eq. (6) using local coordinates, as explained in Section IV-A. Any numerical integration method, either explicit or implicit, could be used to discretize Eq. (9). 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 [15]. Also, it gives more stable and accurate solutions over long time intervals. Using this rule, Eq. (9) is discretized as

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{h}{2}\boldsymbol{U}_c^{\top}\left(\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}) + \boldsymbol{g}(\boldsymbol{x}_{k+1}, \boldsymbol{u})\right), \tag{14}$$

where h is the integration time step. The value $\boldsymbol{x}_{k+1}$ in Eq. (14) is unknown, but it can be obtained using Eq. (8) as

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{x}_{k+1}) &= \boldsymbol{0}, \\ \boldsymbol{U}_c^{\top}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_c) - \boldsymbol{y}_{k+1} &= \boldsymbol{0}. \end{aligned} \tag{15}$$

Now, both Eq. (14) and Eq. (15) are combined to form

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{x}_{k+1}) &= \boldsymbol{0}, \\ \boldsymbol{U}_c^{\top}(\boldsymbol{x}_{k+1} - \tfrac{h}{2}(\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}) + \boldsymbol{g}(\boldsymbol{x}_{k+1}, \boldsymbol{u})) - \boldsymbol{x}_c) - \boldsymbol{y}_k &= \boldsymbol{0}, \end{aligned} \tag{16}$$

where $\boldsymbol{x}_k$, $\boldsymbol{y}_k$, and $\boldsymbol{x}_c$ are known and $\boldsymbol{x}_{k+1}$ is the unknown to determine. Any Newton method can be used to solve this system, but the Broyden method is particularly adequate since it avoids the computation of the Jacobian of the system at each step. Potra and Yen [15] gave an approximation of this Jacobian that allows finding $\boldsymbol{x}_{k+1}$ in few iterations.

For backward integration, i.e., when extending the RRT with root at $\boldsymbol{x}_g$, the time step h in Eq. (16) is negative. In any case, h is adjusted so that the change in parameter space, $\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\|$, is bounded by δ, with $\delta \ll \rho$. This is necessary to detect the transitions between charts, which can occur either because the next state triggers the creation of a new chart (line 12), or because it is not in the part of the manifold covered by the current chart (line 14) and, thus, it is in the part covered by a neighboring chart (line 15).

Algorithm 2: Generate a guiding state $\boldsymbol{x}_{rand}$.

```
1  SAMPLE(A, T)
   input : The atlas, A, the tree currently extended, T.
   output: A sample on the atlas.
2  repeat
3      c ← RANDOMCHARTINDEX(A, T)
4      y_rand ← RANDOMONBALL(σ)
5  until y_rand ∈ P_c
6  x_rand ← x_c + U_c y_rand
7  RETURN(x_rand)
```

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

```
1  CONNECT(A, T, x_near, x_rand)
   input : An atlas, A, a tree, T, the state from where to extend
           the tree, x_near, and the random sample to be reached,
           x_rand.
   output: The updated tree.
2  d_ref ← ‖x_near − x_rand‖
3  repeat
4      d_new ← ∞
5      foreach u ∈ U do
6          x ← SIMULATEACTION(A, T, x_near, x_rand, u)
7          d ← ‖x − x_rand‖
8          if d < d_new then
9              x_new ← x
10             u_new ← u
11             d_new ← d
12     if x_new ∉ T then
13         T ← ADDACTIONSTATE(T, x_near, u_new, x_new)
14     if d_new ≤ d_ref then
15         d_ref ← d_new
16         x_near ← x_new
17 until d_new > d_ref
18 RETURN(T)
```

Algorithm 4: Simulate an action.

1 SIMULATEACTION($A, T, \boldsymbol{x}_k, \boldsymbol{x}_g, \boldsymbol{u}$)
input : An atlas, A, a tree, T, the state from where to start the simulation, $\boldsymbol{x}_k$, the state to approach, $\boldsymbol{x}_g$, and the action to simulate, $\boldsymbol{u}$.
output: The last state in the simulation.
2 $c \leftarrow$ CHARTINDEX($\boldsymbol{x}_k$)
3 FEASIBLE $\leftarrow$ TRUE
4 $t \leftarrow 0$
5 **while** FEASIBLE **and** $\|\boldsymbol{x}_k - \boldsymbol{x}_g\| > \delta$ **and** $|t| \leq t_m$ **do**
6 $\quad \boldsymbol{y}_k \leftarrow \boldsymbol{\varphi}_c(\boldsymbol{x}_k)$
7 $\quad (\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}, h) \leftarrow$ NEXTSTATE($\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{u}, \boldsymbol{F}, \boldsymbol{U}_c, \delta$)
8 $\quad$ **if** COLLISION($\boldsymbol{x}_{k+1}$) **or** OUTOFWORKSPACE($\boldsymbol{x}_{k+1}$) **then**
9 $\quad\quad$ FEASIBLE $\leftarrow$ FALSE
10 $\quad$ **else**
11 $\quad\quad$ **if** $\|\boldsymbol{x}_{k+1} - (\boldsymbol{x}_c + \boldsymbol{U}_c \boldsymbol{y}_{k+1})\| > \varepsilon$ **or** $\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\| / \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| < \cos(\alpha)$ **or** $\|\boldsymbol{y}_{k+1}\| > \rho$ **then**
12 $\quad\quad\quad$ $c \leftarrow$ ADDCHARTTOATLAS($A, \boldsymbol{x}_k$)
13 $\quad\quad$ **else**
14 $\quad\quad\quad$ **if** $\boldsymbol{y}_{k+1} \notin P_c$ **then**
15 $\quad\quad\quad\quad$ $c \leftarrow$ NEIGHBORCHART($A, c, \boldsymbol{y}_{k+1}$)
16 $\quad\quad\quad$ $t \leftarrow t + h$
17 $\quad\quad\quad$ $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k+1}$
18 RETURN($\boldsymbol{x}_k$)

C. Probabilistic completeness

The planner presented is probabilistically complete. Providing a formal proof of this point would be lengthy, and we only sketch the main arguments. Note that the subset of $\mathcal{X}$ that is parameterized by a partial atlas can be densely sampled using the procedure described in Section V-A. Thus, the proof of probabilistic completeness given in [6] for parametric state spaces also holds within this subset. This implies that our planner will be probabilistically complete if, and only if, it is able to extend the atlas to fully parameterize $\mathcal{X}$. This will certainly be achieved if necessary, since the procedure described in Section IV-A ensures that new charts are generated each time the RRT branches approach the border of the subset of $\mathcal{X}$ parameterized at a given moment. The fact that RRT is biased towards such borders (Sec. V-A) ensures that they will eventually be reached, unless the planning problem has been solved before. As shown in [19], the expansion of the atlas will only stop when the atlas has no border, i.e., when it fully covers $\mathcal{X}$.

VI. Test Cases

The planner has been implemented in C and integrated into the CUIK Suite [21]. We next illustrate its performance in three test cases of increasing complexity (See Fig. 4 and https://youtu.be/yV7bDj5zFUs). The first test case was already used in Section III. It consists of a planar four-bar pendulum with limited motor torque that has to move a load. The robot may need to oscillate several times to move from the stable to the unstable equilibrium states shown in Fig. 1(b). The second test case is a planar five-bar robot

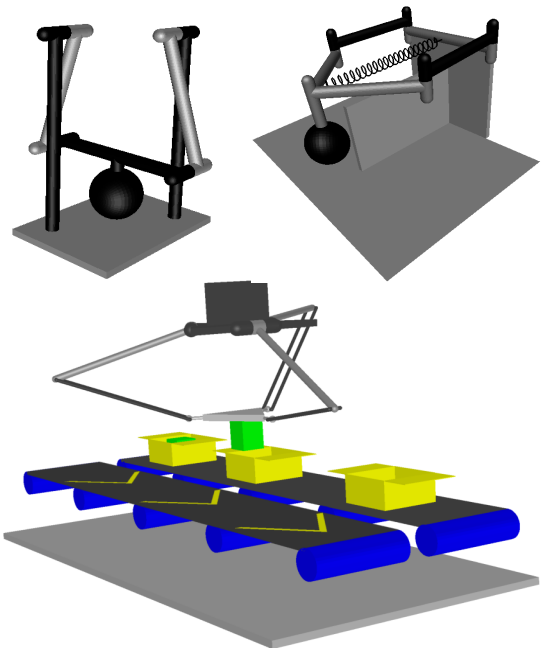

Fig. 4. Test cases used to validate the planner: a four-bar pendulum (top, left), a five-bar robot (top, right), and a Delta robot (bottom).

equivalent to the Dextar prototype [22], but with an added spring to enhance its compliance. The goal here is to move the load from one side to the other of a wall, with null initial and final velocities. Unlike in the first case, collisions with the two walls will easily occur here, and thus they should be avoided. In the third case, a Delta robot moves a heavy load in a pick-and-place scenario, while avoiding obstacles. It picks up the load from a conveyor belt with an initial velocity, and places it at rest inside a box on a second belt. In contrast to typical Delta robot applications, here the weight of the load is considerable, which increases dynamics effects substantially.

In this paper, relative joint angles are used to formulate Eq. (5), while the Euler-Lagrange equations with multipliers are used to formulate Eq. (6).

Table I summarizes the problem dimensions, parameters, and performance statistics of all test cases. As the three robots involve $n_q = 4$, 5, and 15 joints, and each independent kinematic loop introduces 3 or 6 constraints (depending on whether the robot is planar or spatial), the dimensions of the C-space are $d_{\mathcal{C}} = 1$, 2, and 3, respectively. The robots respectively have $n_u = 1$, 2, and 3 of their base joints actuated, while the remaining joints are passive. The set $\mathcal{U}$ is discretized into a finite number of actions, which are randomly chosen at every iteration with uniform distribution between $-\tau_{max}$ and τ_{max}.

In all test cases the parameters are set to $t_m = 0.1$, $\delta = 0.05$, $\sigma = d_{\mathcal{C}}$, $\rho = \sigma/2$, $\cos(\alpha) = 0.1$, and $\varepsilon = 0.1$. Table I gives the value of β which, as in [6], is problem-specific. The table also shows the performance statistics on an iMac with an Intel i7 processor at 2.93 Ghz with 8 CPU cores, which are exploited to run lines 6 to 11 of Algorithm 3

TABLE I

TEST CASE DIMENSIONS, PARAMETERS, AND PERFORMANCE STATISTICS OF THE PLANNER.

Robot	n_q	n_e	$d_{\mathcal{C}}$	$d_{\mathcal{X}}$	No. of actions	τ_{max} [Nm]	β	No. of samples	No. of charts	Plan. Time [s]	t_f [s]
Four-Bars	4	3	1	2	3	16	0.1	272	197	1.0	3.37
	4	3	1	2	3	12	0.1	421	260	1.6	6.30
	4	3	1	2	3	8	0.1	615	388	2.2	5.83
	4	3	1	2	3	4	0.1	1989	1381	7.4	11.42
Five-Bars	5	3	2	4	5	0.2	0.25	6306	291	9.2	32.90
Delta	15	12	3	6	7	1	0.5	1670	83	17.4	8.79

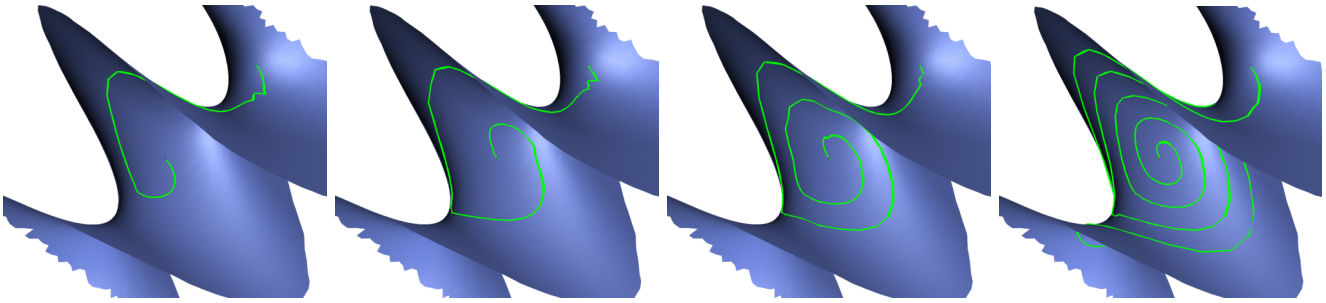

Fig. 5. From left to right, the state manifold (in blue), and the trajectory obtained (green) for a maximum torque τ_{max} of 16, 12, 8, and 4 [Nm].

in parallel. The statistics include the number of samples and charts, as well as the planning time and the trajectory time t_f in seconds, all averaged over ten runs. The planner successfully connected the starting and goal states in all runs. For the sake of comparison, we tried to solve the same test cases with the RRT method in [6], but the planner didn't succeed in any of them.

In the case of the four-bar mechanism, results are included for decreasing values of τ_{max}. As reflected in Table I, the lower the torque, the harder the planning problem. The solution trajectory on the state-space manifold (projected in one position and two velocity variables) can be seen in Fig. 5 for the different values. Clearly, the number of oscillations needed to reach the goal is successively higher. The trajectory obtained for the most restricted case is shown in Fig. 6, top.

In the five-bars robot, although it only has one more link than the previous robot, the planning problem is significantly more complex. This is due to the narrow corridor created by the obstacle to be overcome. Moreover, the motors have a severely limited torque taking into account the spring constant. In order to move the load in such conditions, the planner is forced to increase the momentum of the load before overpassing the obstacle, and to decrease it once it has passed it so as to reach the goal configuration with zero velocity (Fig. 6, middle). This increased complexity is reflected by the high ratio between the number of samples and charts. This shows that, from a given point, the planner has charted the whole of $\mathcal{X}_{\text{free}}$ and, after that, is trying to find a way through the narrow corridor.

Finally, the table gives the same statistics for the problem on the Delta robot. The planning time is higher because the robot is spatial and it has a state space of dimension 6 in an ambient space of dimension 30. Moreover, it involves more holonomic constraints than in the previous cases, and has to avoid collisions with itself and with the environment (the conveyor belts, the boxes and the supporting structure). Also, given the velocity of the belt, the planner is forced to reduce the initial momentum of the load before it can place it inside the box (Fig. 6, bottom).

VII. CONCLUSIONS

This paper has proposed an RRT planner for dynamical systems subject to holonomic constraints. Dealing with such constraints presents two major hurdles: the generation of random samples in the state space and the driftless simulation over such space. We have seen that both issues can be addressed by relying on local parameterizations. The result is a planner that navigates the state space manifold following the vector fields defined by the dynamic constraints on such manifold. The proposed method can successfully solve significantly complex problems.

To scale to even more complex problems, several aspects of the proposed RRT planner need to be improved. Probably the main issue is the metric used to measure the distance between states. This is a general issue of all sampling-based kinodynamic planners, but in our context it is harder since the metric should not only consider the vector fields defined by the dynamic constraints, but also the curvature of the state space manifold defined by the holonomic constraints. Using a metric derived from geometric insights provided by the kinodynamic constraints might result in substantial performance improvements. Also, global optimization methods should be developed to obtain trajectories involving minimum-time or energy consumption [23], [24].

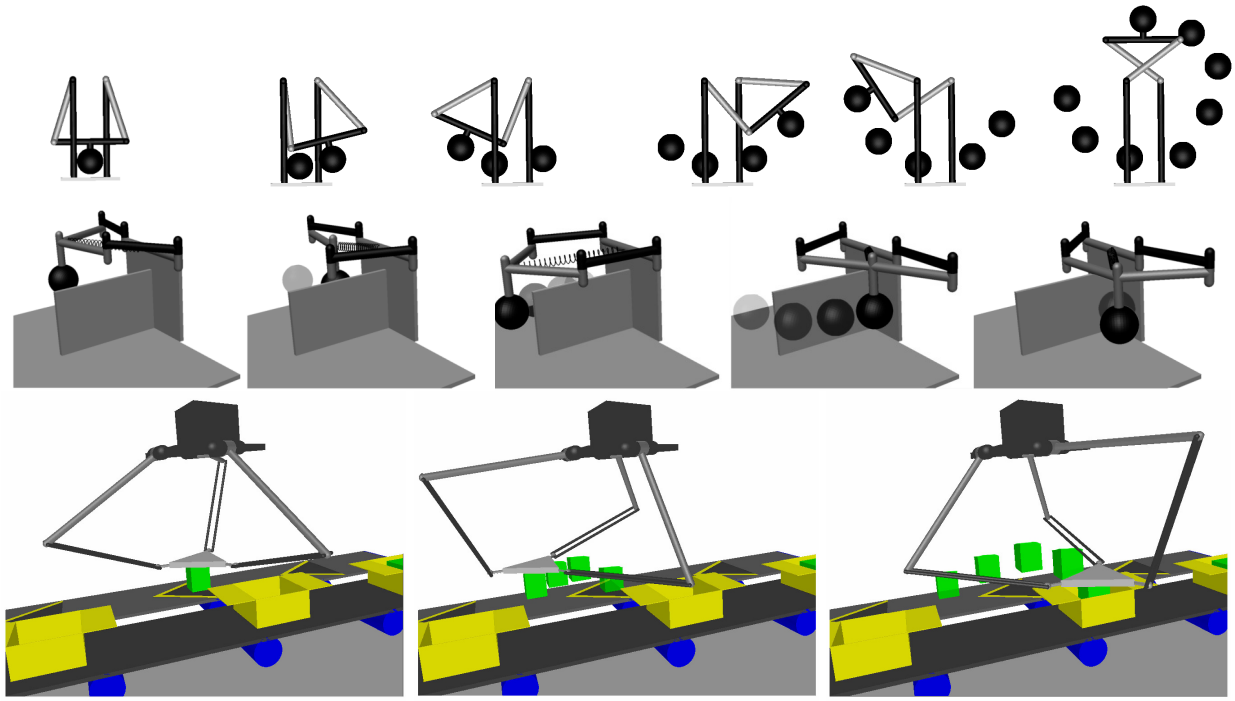

Fig. 6. Snapshots of the trajectories obtained by the planner for the three test cases. See https://youtu.be/yV7bDj5zFUs.

REFERENCES

[1] J.-C. Latombe, *Robot Motion Planning*, ser. Engineering and Computer Science. Springer, 1991.

[2] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *IEEE International Conference on Robotics and Automation*, vol. 1, 2000, pp. 521–528.

[3] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

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

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

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

[7] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[8] L. Jaillet, J. Hoffman, J. van den Berg, P. Abbeel, J. M. Porta, and K. Goldberg, "EG-RRT: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2646–2652.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.