Tangent bundle RRT: A randomized algorithm for constrained motion planning

Beobkyoon Kim, Terry Taewoong Um, Chansu Suh and F. C. Park

Robotica / Volume 34 / Issue 01 / January 2016, pp 202 - 225
DOI: 10.1017/S0263574714001234, Published online: 28 May 2014

Robotica (2016) volume 34, pp. 202–225. © Cambridge University Press 2014
doi:10.1017/S0263574714001234

Tangent bundle RRT: A randomized algorithm for constrained motion planning

Beobkyoon Kim, Terry Taewoong Um, Chansu Suh and F. C. Park*

Robotics Laboratory, Seoul National University, Seoul 151-744, Korea

(Accepted April 21, 2014. First published online: May 28, 2014)

SUMMARY
The Tangent Bundle Rapidly Exploring Random Tree (TB-RRT) is an algorithm for planning robot motions on curved configuration space manifolds, in which the key idea is to construct random trees not on the manifold itself, but on tangent bundle approximations to the manifold. Curvature-based methods are developed for constructing tangent bundle approximations, and procedures for random node generation and bidirectional tree extension are developed that significantly reduce the number of projections to the manifold. Extensive numerical experiments for a wide range of planning problems demonstrate the computational advantages of the TB-RRT algorithm over existing constrained path planning algorithms.

KEYWORDS: Motion planning; Path planning; Control of robotic systems; Redundant manipulators; Service robots.

1. Introduction

The rapidly exploring random tree (RRT)[8,9] has become an essential component of many randomized motion planning algorithms, and several recent efforts have been directed toward developing RRT-based manipulation planning algorithms for kinematically constrained motions.[1,3,13,16] Examples of manipulation tasks involving kinematic constraints are replete in everyday life—using two arms to grasp, lift, and slide a heavy box onto and across a table, for example—and the ability to plan such motions is essential in order for robots to successfully operate in unstructured environments.

What makes the extension of the classical RRT algorithm to constrained motions nontrivial is that the configuration space is no longer a flat space, but typically a curved space. For example, two six degree-of-freedom open chains grasping a common object would have as configuration space a six-dimensional surface embedded in $\mathbb{R}^{12}$. Placing obstacles throughout its workspace would restrict the collision-free portion of the configuration space even further.

Our focus in this paper will be on motion planning problems subject to kinematic equality constraints of the holonomic type, e.g., loop closure equality constraints that characterize closed chain mechanisms, or pose constraints on a manipulator's end-effector. In the presence of such constraints, portions of the configuration space that can be represented by a single multi-dimensional surface will be referred to as a **constraint manifold**. The overall configuration space can consist of a single constraint manifold, as in the example of the two cooperating arms, or may consist of several constraint manifolds of different dimension that may or may not be connected. Examples of planning algorithms for such systems include, e.g.,.[4,12,14]

One of the key steps in the basic RRT algorithm involves the generation of a random sample on the configuration space. Random sampling on vector spaces is straightforward. For our problem, however, the configuration space is no longer a vector space, but a curved manifold embedded in a vector space. Even if the global structure of the curved configuration space were known *a priori*—which it usually is not—generating a random sample in such circumstances would be

* Corresponding author. E-mail: fcp@snu.ac.kr

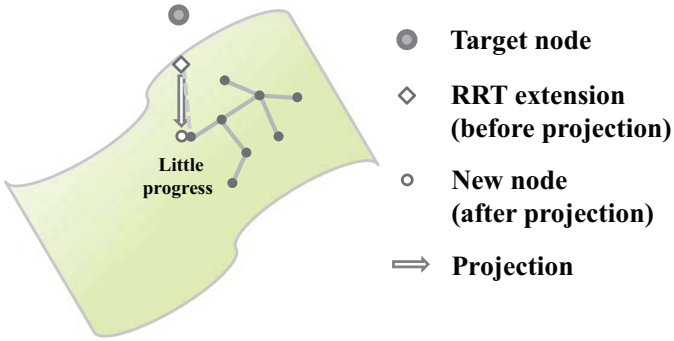


Fig. 1. Example of a target node whose projection contributes very little to extending the tree in the constraint manifold.

time-consuming and computationally intensive. Sampling in a geometrically correct way would require the construction of the associated probability distribution on the curved space in a coordinate-invariant way, among other things.

Methods for constrained motion planning that plan a path directly in some space of reduced dimension (e.g., on a task space constraint manifold), and then attempt to follow this path in the full configuration space (Koga *et al.*,[7] Yamane *et al.*[17]) potentially suffer from feasibility problems—the lower-dimensional path may not be trackable because of joint limits or collisions. Projection methods in configuration space on the other hand seem to have been more successful. For example, Sentis and Khatib[11] use recursive null-space projections to project a robot's configuration away from obstacles while achieving some desired task (although Berenson *et al.*[1] point out that the lack of prior knowledge of the configuration space precludes the use of such task space control techniques as a complete solution).

In the context of RRT methods for motion planning, the basic idea behind projection methods is to randomly sample a point in the ambient Euclidean space, and then to project this point to the configuration space manifold in some appropriate fashion. Yakey *et al.*[16] propose a Randomized Gradient Descent (RGD) algorithm for holonomic closed chains; it should be noted that these configuration spaces typically consist of a single constraint manifold of fixed dimension. Stilman[13] has observed that the RGD method requires considerable parameter tuning and may sometimes fail to satisfy the given constraints, and that in general it is less efficient than, e.g., Jacobian pseudo-inverse projection methods.

One of the more successful general constrained motion planners based on RRTs has been the Constrained Bi-directional Rapidly-Exploring Random Tree (CBiRRT) planner of Berenson *et al.*[1] This method can also be classified as a projection method, and handles a wide range of constraints ranging from loop closure conditions to pose constraints and static torque limits. The CBiRRT algorithm first randomly samples a point q_{target} in the ambient Euclidean space. The algorithm then iteratively moves toward q_{target}, with each step toward q_{target} projected as necessary to the closest constraint manifold. The algorithm has been successfully demonstrated on a wide range of problems, including the dumbbell sliding example described earlier.

Obviously the number of projections, as well as the computational efficiency of the projection operation, are important factors in projection-based planning methods like the CBiRRT algorithm, and reducing the number of projection steps is quite often (though not always) helpful. Less obvious, but possibly even more critical, is the need to consider the shape of the constraint manifold before generating random samples. Figure 1 provides a good illustration of how algorithm performance can be severely diminished if one does not explicitly consider the shape of the constraint manifold when generating random samples. Moreover, irrespective of the particular choice of projection method, the large number of projections of the random configuration to the constraint manifold can significantly impact the performance of the algorithm, both in terms of computational efficiency and the Voronoi bias property of RRT algorithms (essentially, the tendency of RRT algorithms to seek out unexplored regions, see Lindermann and LaValle[20]).

In this paper we propose a new planning algorithm for constrained motions, the Tangent Bundle RRT (TB-RRT) algorithm. By approximating the constraint manifold as a collection of tangent spaces—which are vector spaces, so that standard RRT algorithms can be straightforwardly applied to each tangent space—the RRT algorithm can be generalized to constraint manifolds in a way that preserves the defining features of the RRT-based planning (computational efficiency, Voronoi bias property), while offering significant improvements in computational efficiency over existing constrained planning algorithms. More specifically, whereas the CBiRRT algorithm samples nodes in the ambient Euclidean space, and immediately projects this node to the constraint manifold, the key idea in the TB-RRT algorithm is to first sample and construct an RRT in the tangent bundle, i.e., the collection of all tangent spaces to the constraint manifold.

The idea of sampling in the tangent space has also been examined in ref. [16], but there the random node is projected immediately back to the constraint manifold. In our approach, only when the RRT constructed in a particular tangent space reaches a boundary—each of the tangent spaces are bounded—or deviates from the constraint manifold by a certain prescribed threshold, does the algorithm project back to the constraint manifold, at which point the procedure is repeated, i.e., creating another bounded tangent space, constructing an RRT, and reprojecting. The efficiency of our algorithm is further improved by the following features:

- Local curvature information is used to *a priori* "size" each tangent space, by using larger tangent spaces in regions of lower curvature;
- Heuristics are developed for choosing the initial RRT construction direction so as to maintain the Voronoi bias property toward unexplored regions;
- Like,[1] our algorithm is also bidirectional in that it generates trees from both the start and goal nodes;
- A "lazy projection" procedure similar to the lazy collision checking procedure of Sanchez and Latombe[10] is developed to improve the efficiency of the projection step.

Because of the increased preprocessing and bookkeeping, it is a valid question whether the TB-RRT algorithm will perform better than, e.g., the CBi-RRT algorithm, whose computational procedure is much simpler but generally requires a greater number of iterations to generate a feasible path. Our case studies confirm that even with the additional preprocessing and bookkeeping, the TB-RRT algorithm outperforms existing constrained motion planners for a large class of benchmark problems; results of our empirical studies suggest that the performance improvements offered by the TB-RRT algorithm are greater for more complex planning problems.

The TB-RRT algorithm is an outgrowth of earlier preliminary work by the authors first presented in refs. [15, 21, 22]. Whereas the idea of constructing random trees on the tangent bundle is first presented in these papers, the current paper goes further in a number of substantive ways: we derive explicit curvature formulas that are used to bound each tangent space according to the local curvature of the constraint manifold; we develop a more systematic set of rules for, e.g., generating new nodes and lazy projection; we perform a more extensive set of numerical experiments for more complex planning environments, comparing our results with several alternative constrained planning algorithms.

One of the alternative constrained planning algorithms that we compare is the higher-dimensional continuation method of ref. [23, 24]; these methods adopt the tangent bundle framework first presented in ref. [15], but make use of numerical continuation methods[5] for constructing local coordinate charts. In ref. [24] the sampling domain on the tangent space is initially set to be a ball of some pre-specified radius R; linear inequality constraints are then applied to this ball domain for each neighboring tangent space that intersects. The resulting *AtlasRRT* algorithm constructs coordinate charts in a minimally overlapping way, but involves, among other things, repeated computation of intersections of volumes (usually open balls) and other computationally intensive tasks. A performance comparison between the AtlasRRT algorithm and our curvature-based TB-RRT algorithm shows that our approach uniformly outperforms AtlasRRT, in some cases by orders of magnitude.

The paper is organized as follows. In Section 2 we review the geometry of the tangent bundle, and provide formulas for orthogonal projections to the tangent and normal subspaces. We also explicitly derive, in local coordinates, formulas for the principal curvatures and principal directions of the constraint manifold; these formulas should be of independent interest for other applications besides planning. Section 3 describes the complete version of the TB-RRT algorithm. Section 4 describes

a simpler variation of the TB-RRT algorithm[1], in which random nodes generated in the ambient configuration space are projected to the nearest tangent space. Experimental results are presented in Section 5. Conclusions and further extensions of our algorithm are presented in Section 6.

2. Geometry of the Tangent Bundle

This section describes the tangent bundle structure of the constraint manifold. After defining the constraint manifold and its associated tangent bundle, formulas are given for orthogonal projection into the tangent and normal subspaces of the constraint manifold. We also derive, in local coordinates, explicit formulas for the principal curvatures and principal directions of the constraint manifold (in the literature, the treatment of curvature of Riemannian manifolds is with few exceptions done in a coordinate-free manner; in this regard our formulas should also be of independent interest).

The motion planning problem addressed in this paper can be characterized mathematically as follows. Let $\mathcal{M}$ denote the configuration space, or C-space, of the robot system, and $u \in \mathbb{R}^m$ denote a set of local coordinates for $\mathcal{M}$. $\mathcal{M}$ is assumed to have the structure of a differentiable manifold (usually with boundary—to simplify the exposition we shall not concern ourselves with the technical details of manifolds with boundaries, since for the most part generalization to boundaries should be intuitively clear). In many cases, particularly robots that form closed loops, $\mathcal{M}$ itself can be characterized as a submanifold of a higher-dimensional space $\mathcal{X}$ of dimension $n > m$, with local coordinates $q \in \mathbb{R}^n$.

Denoting coordinates for $\mathcal{X}$ by q, for the purposes of this paper the configuration space of a robot system is assumed to be implicitly characterized by algebraic equality constraints of the form $f(q) = 0$. The set of all feasible configurations q then forms a submanifold of $\mathcal{X}$, which we refer to as the constraint manifold $\mathcal{M}$. To better describe our algorithm, we shall not explicitly consider inequality constraints of the form $h(q) \geq 0$ here; such collision avoidance-type constraints can be incorporated into our planning framework with only modest effort, but the resulting algorithm structure and description becomes considerably more involved.

2.1. Manifolds, tangent bundles, and projections

For our purposes it is sufficient to consider manifolds $\mathcal{M}$ of dimension m that are embedded in $\mathbb{R}^n$, with $m \leq n$; $\mathcal{M}$ can then be visualized as an m-dimensional surface in $\mathbb{R}^n$. Denote by $u \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$ local coordinates for $\mathcal{M}$ and $\mathbb{R}^n$, respectively. Any point x lying on $\mathcal{M}$ is then locally parametrized by u, i.e., $x = x(u)$. x is a regular point of $\mathcal{M}$ if the matrix

$$\frac{\partial x}{\partial u}(u) = \begin{bmatrix} \frac{\partial x_1}{\partial u_1} & \cdots & \frac{\partial x_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial u_1} & \cdots & \frac{\partial x_n}{\partial u_m} \end{bmatrix} \in \mathbb{R}^{n \times m}$$

is of full rank m; otherwise x is a singular point of $\mathcal{M}$. If $u(t) \in \mathbb{R}^m$ is a differentiable curve, then $x(t) = x(u(t))$ is a differentiable curve on $\mathcal{M}$; differentiating with respect to t leads to $\dot{x} = (\partial x/\partial u)\dot{u}$. At a regular point $x = x(u)$, $\partial x/\partial u$ is by definition full rank, implying that its m columns are linearly independent; the m-dimensional subspace of $\mathbb{R}^n$ spanned by these m linearly independent columns then defines the **tangent space** to $\mathcal{M}$ at x, denoted $T_x\mathcal{M}$. The subspace of $\mathbb{R}^n$ that is normal to $T_x\mathcal{M}$ is called the **normal subspace**, and denoted $T_x\mathcal{M}^\perp$. The collection of all points $x \in \mathcal{M}$, together with the tangent spaces $T_x\mathcal{M}$, is then called the **tangent bundle** of $\mathcal{M}$, and denoted $T\mathcal{M}$.

$T_x\mathcal{M}$ is a vector space of dimension m, and the Euclidean inner product on $\mathbb{R}^n$ can be used to define an inner product on $T_x\mathcal{M}$ as follows. Measuring incremental arclength on $\mathbb{R}^n$ as $ds^2 = dx_1^2 + \ldots + dx_n^2$, it follows that

$$ds^2 = \sum_{i=1}^{m} \sum_{j=1}^{m} g_{ij}(u) du_i du_j,$$

[1] A preliminary version of this simpler variation of TB-RRT was reported in ref. [22]

where $g_{ij}(u)$ is the (i, j) entry of the $m \times m$ matrix

$$G(u) = \left(\frac{\partial x}{\partial u}\right)^T \left(\frac{\partial x}{\partial u}\right). \tag{1}$$

$G(u) = (g_{ij})$ is the **Riemannian metric** on $\mathcal{M}$ induced by the Euclidean metric on $\mathbb{R}^n$, and can equivalently be regarded as the first fundamental form of $\mathcal{M}$.

Given any $v \in \mathbb{R}^n$ with its base situated at $x \in \mathcal{M}$—we do not require that $v \in T_x\mathcal{M}$ here—the linear map $P : \mathbb{R}^n \to \mathbb{R}^n$, defined by

$$P = \frac{\partial x}{\partial u} G^{-1}(u) \frac{\partial x}{\partial u}^T, \tag{2}$$

where $(\partial x/\partial u)$ and G^{-1} are evaluated at u corresponding to $x = x(u)$, then orthogonally projects v to the tangent space; that is, $Pv \in T_x\mathcal{M}$. If initially $v \in T_x\mathcal{M}$, then $Pv = v$.

Given any $v \in \mathbb{R}^n$ with its base situated at $x \in \mathcal{M}$ as before, the linear mapping $N : \mathbb{R}^n \to \mathbb{R}^n$ defined by

$$N = I - P \tag{3}$$

then orthogonally projects v to the normal subspace; that is, $Nv \in T_x\mathcal{M}^\perp$. If initially $v \in T_x\mathcal{M}^\perp$, then $Nv = v$.

The above formulas are derived in terms of an explicit local coordinate representation $x = x(u)$ of $\mathcal{M}$. Manifolds can also be characterized implicitly by an equation of the form $f(x) = 0$, where $f : \mathbb{R}^n \to \mathbb{R}^{n-m}$ is differentiable. If the $(n - m) \times n$ matrix

$$\frac{\partial f}{\partial x}(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n-m}}{\partial x_1} & \cdots & \frac{\partial f_{n-m}}{\partial x_n} \end{bmatrix} \tag{4}$$

is of full rank $n - m$ at x, then x is a regular point of $\mathcal{M}$. An important special case is when $n = m + 1$; in this case $\mathcal{M}$ is said to be a **hypersurface** in $\mathbb{R}^n$, characterized by the scalar implicit equation $f(x_1, \ldots, x_n) = 0$.

If $x(t)$ is a curve lying on $\mathcal{M}$, then differentiating both sides of $f(x(t)) = 0$ leads to

$$\frac{\partial f}{\partial x}\dot{x} = 0.$$

Since by definition $\dot{x} \in T_x\mathcal{M}$, it follows that if x is a regular point, then the $n - m$ rows of $(\partial f/\partial x)(x)$ span the normal subspace $T_x\mathcal{M}^\perp$. For convenience partition x into $x = (u, v)$, where $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^{n-m}$, and assume that the $(n - m) \times (n - m)$ matrix $(\partial f/\partial v)$ is invertible at x. Then u can be regarded as a set of local coordinates for $\mathcal{M}$. Letting $x(t) = (u(t), v(t))$ be some arbitrary differentiable curve on $\mathcal{M}$ and differentiating both sides of $f(x(t))f(u(t), v(t)) = 0$ leads to

$$\frac{\partial f}{\partial u}\dot{u} + \frac{\partial f}{\partial v}\dot{v} = 0, \tag{5}$$

so that we can write

$$dv = -\left(\frac{\partial f}{\partial v}\right)^{-1} \frac{\partial f}{\partial u}\, du. \tag{6}$$

The metric $ds^2 = dx_1^2 + \ldots dx_n^2$ then becomes

$$\begin{aligned} ds^2 &= du_1^2 + \ldots + du_m^2 + dv_1^2 + dv_{n-m}^2 \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m} g_{ij}(u) du_i du_j, \end{aligned}$$

where $g_{ij}(u)$ is the (i, j) entry of the following matrix $G(u)$:

$$G(u) = I + \left(\left(\frac{\partial f}{\partial v}\right)^{-1}\frac{\partial f}{\partial u}\right)^T \left(\left(\frac{\partial f}{\partial v}\right)^{-1}\frac{\partial f}{\partial u}\right). \tag{7}$$

2.2. Curvature formulas

Before deriving the principal curvature formulas for multidimensional surfaces, it is instructive to review this notion for two-dimensional surfaces in $\mathbb{R}^3$. Let $u \in \mathbb{R}^2$ denote local coordinates for a surface $\mathcal{S}$ embedded in $\mathbb{R}^3$, and $x(u) \in \mathcal{S}$ denote a point on the surface. The first fundamental form is then given by

$$G(u) = \frac{\partial x}{\partial u}^T \frac{\partial x}{\partial u}.$$

The two column vectors of $\partial x/\partial u$, denoted

$$\frac{\partial x}{\partial u_1} = \begin{bmatrix} \frac{\partial x_1}{\partial u_1} \\ \frac{\partial x_2}{\partial u_1} \\ \frac{\partial x_3}{\partial u_1} \end{bmatrix}, \quad \frac{\partial x}{\partial u_2} = \begin{bmatrix} \frac{\partial x_1}{\partial u_2} \\ \frac{\partial x_2}{\partial u_2} \\ \frac{\partial x_3}{\partial u_2} \end{bmatrix},$$

are both tangent to $\mathcal{S}$. The unit normal to $\mathcal{S}$ can be obtained as

$$\mathfrak{n} = \left(\frac{\partial x}{\partial u_1} \times \frac{\partial x}{\partial u_2}\right) \Big/ \left\| \left(\frac{\partial x}{\partial u_1} \times \frac{\partial x}{\partial u_2}\right) \right\|.$$

Now consider an arbitrary curve $x(t)$ lying on $\mathcal{S}$. Clearly the tangent vector $\dot{x}$ will always be normal to $\mathfrak{n}$, i.e., $\langle \dot{x}, \mathfrak{n} \rangle = 0$, where $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product in $\mathbb{R}^3$. For curved surfaces $\langle \ddot{x}, \mathfrak{n} \rangle$ is nonzero, and in fact measures the degree to which the surface is curved along $\dot{x}$. Calculating $\langle \ddot{x}, \mathfrak{n} \rangle$ explicitly,

$$\langle \ddot{x}, \mathfrak{n} \rangle = \left\langle \frac{\partial^2 x}{\partial u_1^2}, \mathfrak{n} \right\rangle \dot{u}_1^2 + 2\left\langle \frac{\partial^2 x}{\partial u_1 \partial u_2}\mathfrak{n} \right\rangle \dot{u}_1 \dot{u}_2 + \left\langle \frac{\partial^2 x}{\partial u_2^2}, \mathfrak{n} \right\rangle \dot{u}_2^2.$$

$\langle \ddot{x}, \mathfrak{n} \rangle$, which is the projection of $\ddot{x}$ in the normal direction $\mathfrak{n}$, is therefore quadratic in $\dot{u}_1$ and $\dot{u}_2$. The resulting quadratic form is the second fundamental form of the surface, which can also be written

$$\langle \ddot{x}, \mathfrak{n} \rangle \, dt^2 = \sum_{i=1}^{2}\sum_{j=1}^{2} h_{ij}(u) du_i du_j,$$

where

$$H(u) = \begin{bmatrix} h_{11}(u)\ h_{12}(u) \\ h_{21}(u)\ h_{22}(u) \end{bmatrix} = \begin{bmatrix} \left\langle \frac{\partial^2 x}{\partial u_1^2}, \mathfrak{n} \right\rangle & \left\langle \frac{\partial^2 x}{\partial u_1 \partial u_2}, \mathfrak{n} \right\rangle \\ \left\langle \frac{\partial^2 x}{\partial u_1 \partial u_2}, \mathfrak{n} \right\rangle & \left\langle \frac{\partial^2 x}{\partial u_2^2}, \mathfrak{n} \right\rangle \end{bmatrix}.$$

The principal curvatures and principal directions of the surface are then given by the eigenvalues and eigenvectors of $G^{-1}H$, respectively. At any given point x on the surface, the principal directions, recall, indicate the directions of maximum and minimum curvature.

Principal curvatures and directions for multidimensional surfaces can also be defined following more or less the same procedure (with one important difference—the surface normal may not be unique). As before let $\mathcal{M}$ denote an m-dimensional surface embedded in $\mathbb{R}^n$, $u \in \mathbb{R}^m$ be local coordinates for $\mathcal{M}$, and $x(u) \in \mathcal{M}$ a point on $\mathcal{M}$. Let $G(u)$ be the induced metric on $\mathcal{M}$, and $g_{ij}(u)$ be the (i, j) element of G. Further define

$$x_{ij} = \frac{\partial^2 x}{\partial u_i \partial u_j}, \tag{8}$$

and let $\langle \cdot, \cdot \rangle$ denote the Euclidean inner product in $\mathbb{R}^n$. Let $\mathfrak{n} \in \mathbb{R}^n$ be a unit normal to $\mathcal{M}$ at x. The second fundamental form with respect to $\mathfrak{n}$, denoted $Q_{\mathfrak{n}}$, is an $m \times m$ matrix whose (i, j) element is given by $\langle \mathfrak{n}, x_{ij} \rangle$. The principal curvatures and directions of $\mathcal{M}$ are respectively given by the eigenvalues and eigenvectors of $G^{-1}Q_{\mathfrak{n}}$.

Unless $\mathcal{M}$ is a hypersurface (that is, a surface of dimension m embedded in $\mathbb{R}^{m+1}$), the choice of $\mathfrak{n}$ will not be unique. In this case a popular choice is to choose the mean curvature vector, defined as follows:

$$v = \frac{1}{m} N \sum_{i=1}^{m} \sum_{j=1}^{m} g^{ji} x_{ij}, \tag{9}$$

where $N \in \mathbb{R}^{n \times n}$ is the orthogonal projection operator to the normal subspace $T_x\mathcal{M}^{\perp}$ that was defined earlier, and g^{ji} denotes the (j, i) element of G^{-1}. v can be interpreted as a (coordinate-invariant) average of the x_{ij}'s that is projected to $T_x\mathcal{M}^{\perp}$. Choosing the surface normal $\mathfrak{n}$ to be $v/\|v\|$, the (i, j) element of the second fundamental form $Q_{\mathfrak{n}}$ is given by

$$\left\langle x_{ij}, \frac{1}{m\|\mathfrak{n}\|} \sum_{k=1}^{m} \sum_{l=1}^{m} g_{lk} N x_{kl} \right\rangle. \tag{10}$$

The principal curvatures and directions are then given by the eigenvalues and eigenvectors of $G^{-1}Q_{\mathfrak{n}}$.

We end this section by working out the corresponding curvature formulas when the constraint manifold is represented implicitly in the form $f(x) = 0$. As before, partition x into (u, v), $u \in \mathbb{R}^m$, $v \in \mathbb{R}^{n-m}$, in such a way that $\partial f/\partial v$ is invertible. The first fundamental form $G(u)$ is then given by (7). Note that

$$\frac{\partial x}{\partial u} = \begin{bmatrix} I \\ -\left(\frac{\partial f}{\partial v}\right)^{-1} \frac{\partial f}{\partial u} \end{bmatrix}, \tag{11}$$

where I is the $m \times m$ identity matrix. Since $\partial x/\partial u_j$ is by definition the j-th column of $\partial x/\partial u$, it follows that

$$\frac{\partial^2 x}{\partial u_i \partial u_j} = \begin{bmatrix} 0 \\ \dfrac{\partial}{\partial u_i}\left(-\left(\dfrac{\partial f}{\partial v}\right)^{-1}\dfrac{\partial f}{\partial u_j}\right) \end{bmatrix},$$

which simplifies to

$$\begin{bmatrix} 0 \\ \left(\dfrac{\partial f}{\partial v}\right)^{-1}\frac{\partial^2 f}{\partial u_i \partial v}\left(\dfrac{\partial f}{\partial v}\right)^{-1}\frac{\partial f}{\partial u_j} - \left(\dfrac{\partial f}{\partial v}\right)^{-1}\dfrac{\partial^2 f}{\partial u_i \partial u_j} \end{bmatrix}$$

by use of the general matrix identity

$$\frac{d}{dt}A^{-1}(t) = -A^{-1}\left(\frac{d}{dt}A(t)\right)A^{-1}.$$

3. Tangent Bundle RRT Algorithm

3.1. Main steps of algorithm

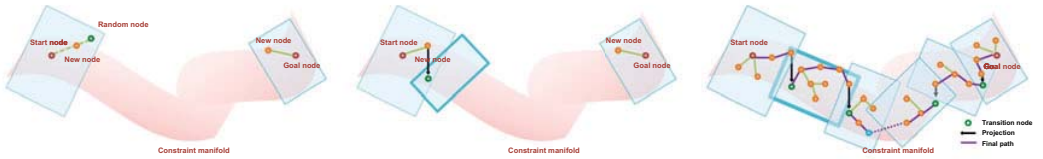
The basic RRT algorithm introduced in ref. [9] is shown in Algorithm 1). Taking this algorithm as our point of departure, the TB-RRT algorithm is described as an extension of the basic RRT algorithm.

Algorithm 1 RRT_CONNECT_PLANNER(q_{init}, q_{goal})

```
1: Tree_a.init(q_init); Tree_b.init(q_goal)
2: for k = 1 to K do
3:    q_rand ← RANDOM_COFIG();
4:    if not (EXTEND(Tree_a, q_rand) = Trapped) then
5:       if (CONNECT(Tree_b), q_new) = Trapped) then
6:          Return PATH(Tree_a, Tree_b);
7:       end if
8:    end if
9:    SWAP(Tree_a, Tree_b);
10: end for
11: Return Failure
```

The main distinguishing feature of TB-RRT is that branches of exploring trees are constructed on the tangent spaces of the constraint manifold instead of in the configuration space. Assuming that the initial and final configurations, respectively denoted by q_{init} and q_{final}, are already on the constraint manifold $\mathcal{M}$, and that two tangent spaces to $\mathcal{M}$ have been constructed at q_{init} and q_{final}, the algorithm proceeds sequentially as follows:

1. Using the selection bias function described in Section 3.7, a tangent space among the existing collection of tangent spaces is first selected.
2. A random node q_{rand} is generated on the selected tangent space, and the nearest neighbor node q_{near} lying on the same tangent space is found.
3. The tree is extended from q_{near} toward q_{rand}. Our RRT methods can be categorically classified as RRT-ExtCon or RRT-ConCon.[9] The only difference between the two modes is in the extension phase, which we describe separately below.
   (a) RRT-ExtCon: The tree is extended by a single step of fixed stepsize. Denote this extended node by q_{new}. q_{new} is tested to see if the inequality constraints are satisfied. If it fails to satisfy the inequality constraints, the node is abandoned. If it satisfies the inequality constraints, then the distance from q_{new} to the constraint manifold $\mathcal{M}$ is determined. If this distance exceeds a



(a) Generate a random sample node on a tangent space, and extend the tree on the tangent space.

(b) When the distance $\|e\|$ from the extended node q_{new} to $\mathcal{M}$ exceeds a certain prescribed distance $E_{\mathcal{M}}$, the node is projected onto the manifold, and a new bounded tangent space is created. Therefore, multiple tangent spaces could be created as the random trees are exploring the constraint manifold.

(c) The final path is extracted through multiple tangent spaces. The actual path on the manifold can be obtained by projecting the nodes on the final path.

Fig. 2. Illustration of the TB-RRT algorithm.

prescribed threshold $E_{\mathcal{M}}$, then q_{new} is projected onto $\mathcal{M}$; this projected point is then designated as the new q_{new}, and a new tangent space is created at this new point on $\mathcal{M}$ (see Algorithm 3).

(b) RRT-ConCon: The tree is extended as much as possible until it reaches q_{rand}. If the extended node fails to satisfy the inequality constraints, the node is abandoned; the last extended node is designated q_{new}, and the extension phase stops. If the extended node satisfies the inequality constraints, then the distance from q_{new} to the constraint manifold $\mathcal{M}$ is determined. If the distance exceeds a prescribed threshold $E_{\mathcal{M}}$, then the node is projected onto $\mathcal{M}$; this projected point is designated q_{new}, and a new tangent space is created at this new point on $\mathcal{M}$; the extension phase stops (see Algorithm 4).

4. If there is no extended node, the above procedure is repeated from the selection of an existing tangent space.
5. An attempt is made to connect q_{new} with the opposite tree. If successful, a final path has been found and the algorithm terminates.
6. Otherwise, the nearest neighbor node to q_{new} on the opposite tree is found, and q_{new} is projected onto the tangent space on which this nearest neighbor node lies. The projected point becomes the target node for the opposite tree; that is, we apply the same extension procedure described above to the opposite tree, and determine a corresponding q_{new}.
7. An attempt is made to connect q_{new} with the opposite tree. If successful, a final path has been found and the iteration terminates. Otherwise, the algorithm repeats with the selection of an existing tangent space, random sampling on this tangent space, etc., until the algorithm successfully terminates or exceeds the maximum number of iterations.
8. Once a successful connection is established, the final path through the collection of tangent bundles is extracted, and the nodes of the final path are projected onto the constraint manifold (see Algorithm 8).

The pseudo-code description of the TB-RRT algorithm is given in Algorithm 2 (see Fig. 2). We now discuss details of the key steps of our algorithm.

3.2. Projection

Assuming $\mathcal{M}$ is a surface embedded in some higher dimensional normed Euclidean space, given a configuration q not on $\mathcal{M}$, the natural way to compute the distance between q and $\mathcal{M}$ is to find the point $p \in \mathcal{M}$ that minimizes the distance $\|p - q\|$, where $\|\cdot\|$ is a suitably chosen norm on the Euclidean space. Because this nonlinear optimization problem is often difficult and computationally intensive, in practice one settles for easily obtained solutions that approximately minimize the distance function. If the constraint manifold is parametrized implicitly as $f(q) = 0$, one easily implementable optimization procedure is to define the error vector e according to

$$e = f(q), \tag{12}$$

Algorithm 2 TB-RRT(q_{init}, q_{final})

```
1: Tree_a, Tree_b, TangentSpaces[]
2: Tree_a.AddVertex(q_init), Tree_b.AddVertex(q_final)
3: TangentSpaces.Add(CreateTangentSpace(q_init))
4: TangentSpaces.Add(CreateTangentSpace(q_final))
5: while i < I_max do
6:    i ← i + 1
7:    k ← SelectTangentSpace()
8:    Tree_k ← FindTree(k)
9:    Tree_o ← FindOppositeTree(k)
10:   q_rand ← RandomSampleOnTangentSpace(TangentSpaces[k])
11:   q_near ← FindNearestNodeOnTangentSpace(k, q_rand)
12:   if q_new ← Extend(Tree_k, q_near, q_rand) then
13:      if Connect(q_new, Tree_o) then
14:         Path ← ExtractPath()
15:      end if
16:      return LazyProjection(Path)
17:   else
18:      goto 5
19:   end if
20:   q_near ← FindNearestNodeOnTree(Tree_o, q_new)
21:   q_new ← ProjectOntoTangentSpace(q_new, q_near)
22:   if q_new ← Extend(Tree_o, q_near, q_new) then
23:      if Connect(q_new, Tree_k) then
24:         Path ← ExtractPath()
25:         return LazyProjection(Path)
26:      end if
27:   else
28:      goto 5
29:   end if
30: end while
31: return NULL
```

Algorithm 3 Extend(Tree, q_{near}, q_{target}) *(RRT-ExtCon)*

```
1: q_new ← ExtendOneStep(q_near, q_target)
2: if CollisionOccur(q_new) then
3:    return NULL
4: end if
5: if ‖f(q_new)‖ > E_M then
6:    if ProjectionNewtonRaphson(q_new) then
7:       TangentSpaces.Add(CreateTangentSpace(q_new))
8:    else
9:       return NULL
10:   end if
11: end if
12: Tree.AddVertex(q_new), Tree.AddEdge(q_near, q_new)
13: return q_new
```

i.e., if q lies on $\mathcal{M}$ then e is zero, and nonzero otherwise. For q sufficiently close to $\mathcal{M}$, $\|e\|^2$ is a valid distance function that can be easily minimized via a Newton-Raphson root-finding procedure for $f(q)$ (see Algorithm 6). Specifically, a first-order Taylor expansion of (12) leads to

$$e + \delta e \simeq f(q) + \frac{\partial f}{\partial q}(q)\delta q. \tag{13}$$

Algorithm 4 Extend(Tree, q_{near}, q_{target}) *(RRT-ConCon)*

$q_{\text{new}} \leftarrow$ NULL, $q_{\text{old}} \leftarrow q_{\text{near}}$
while ($q_{\text{new}} = q_{\text{target}}$) **do**
  $q_r \leftarrow$ ExtendOneStep(q_{old}, q_{target})
  if CollisionOccur(q_r) **then**
    return q_{new}
  end if
  if $\|f(q_r)\| > E_{\mathcal{M}}$ **then**
    if ProjectionNewtonRaphson(q_r) **then**
      Tree.AddVertex(q_r), Tree.AddEdge(q_{old}, q_r)
      TangentSpaces.Add(CreateTangentSpace(q_r))
      return q_r
    else
      return q_{new}
    end if
  end if
  $q_{\text{new}} \leftarrow q_r$
  Tree.AddVertex(q_{new}), Tree.AddEdge(q_{old}, q_{new})
  $q_{\text{old}} \leftarrow q_{\text{new}}$
end while
return q_{new}

Algorithm 5 ExtendOneStep(q, q_{target})

$q_{\text{ext}} \leftarrow q + \min(\triangle s, \|q_{\text{target}} - q\|)\frac{(q_{\text{target}} - q)}{\|q_{\text{target}} - q\|}$
return q_{ext}

Here we denote the Jacobian matrix $\frac{\partial f}{\partial q}$ by $J(q)$. Setting the right-hand side of (13) to zero and solving for δq, we obtain the update rule

$$q_{new} \leftarrow q - J(q)^{\dagger}\delta e, \tag{14}$$

where $J(q)^{\dagger}$ denotes the pseudo-inverse of $J(q)$:

$$J(q)^{\dagger} = J(q)^{T}[J(q)J(q)^{T}]^{-1}. \tag{15}$$

The projection matrix $P(q)$, evaluated as in (2), is used to project a node onto the tangent space (see Algorithm 7). We remark that the projection occurs only for (i) the root nodes of tangent spaces and (ii) the nodes of the final path.

Algorithm 6 ProjectionNewtonRaphson(q)

while $j < J_{\max}$ **do**
  $j \leftarrow j + 1$
  $e \leftarrow f(q)$
  if $\|e\| < \varepsilon$ **then**
    return q
  end if
  $q \leftarrow q - J(q)^{\dagger}e$
end while
return NULL

Algorithm 7 ProjectionOntoTangentSpace(q, q_a)

1: $P \leftarrow$ GetProjectionMatrix(q_a)
2: $q_p \leftarrow q_a + P(q - q_a)$
3: **return** q_p

Algorithm 8 LazyProjection(Path)

1: **for** $i = 1$, Path.size **do**
2: Path.node[i] $\leftarrow$ ProjectionNewtonRaphson(Path.node[i])
3: **end for**
4: **return** Path

3.3. Random Sampling on Tangent Spaces

Let $\{b_i, \ldots, b_m\}$ be a basis for the tangent space at a root node q_{root} to $\mathcal{M}$. For each $b_i \in \mathbb{R}^n$ we assign a scalar value $r_i \in \mathbb{R}$ to define the range over which the random number is generated. A random sample q_{rand} on the tangent space can then be generated straightforwardly as

$$q_{\text{rand}} = q_{\text{root}} + \sum_{i=1}^{m} w_i b_i \tag{16}$$

where $w_i \in \mathbb{R}$ denotes a random number generated in the range $(-r_i, r_i)$.

3.4. Tangent Space Basis and Boundaries

A basis for each tangent space, as well as its boundaries—recall that our tangent spaces have boundaries—can be obtained in a number of ways. One effective way uses the local curvature information of the constraint manifold. The basic premise is that at root nodes where the principal curvatures are nearly zero, the manifold is nearly flat along the corresponding principal directions, and the tangent space can be extended further along such directions without deviating significantly from the manifold. Conversely, when the principal curvatures are relatively large, the manifold is more highly curved along these corresponding principal directions; it is thus reasonable to restrict the boundary of the tangent space along these directions to minimize the deviation of the tangent space from the manifold.

The first order of business is to compute the principal curvatures and directions as introduced in Section 2: given a basis matrix $\frac{\partial x}{\partial u} \in \mathbb{R}^{n \times m}$ defined in (11), the second fundamental form $Q_{\mathfrak{n}}$ can be evaluated as in (9) and (10). Denote the eigenvalues and eigenvectors of $Q_{\mathfrak{n}}$ by, respectively, $c_i \in \mathbb{R}$ and $d_i \in \mathbb{R}^m$ for $i = 1, \ldots, m$. The principal basis for random sampling purposes can then be computed as

$$b_i = \frac{\partial x}{\partial u} d_i. \tag{17}$$

Note that c_i represents the (squared) principal curvatures in the principal direction b_i. To bound the tangent spaces we adopt the following set of rules. First, for each principal direction b_i, the bound r_i is computed as

$$r_i = \sqrt{2\rho E_{\mathcal{M}} - {E_{\mathcal{M}}}^2} \tag{18}$$

where ρ is determined from the following procedure:

- Initialize $\rho = \frac{1}{\sqrt{c_i}}$;
- If ($\rho < \rho_{\min}$) then $\rho = \rho_{\min}$;
- Else if ($\rho > \rho_{\max}$) then $\rho = \rho_{\max}$;

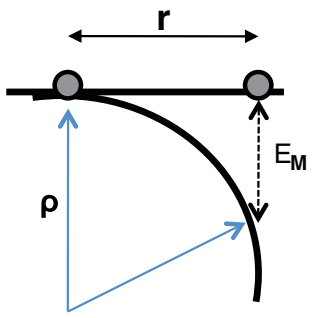


Fig. 3. Example of circular manifold intersection.

where $\rho_{\min}$ and $\rho_{\max}$ are defined as

$$\rho_{\min} = \frac{(\text{stepsize})^2 + {E_{\mathcal{M}}}^2}{2E_{\mathcal{M}}}, \quad \rho_{\max} = \frac{(q_{\text{final}} - q_{\text{init}})^2 + {E_{\mathcal{M}}}^2}{2E_{\mathcal{M}}}. \tag{19}$$

The choice of Eq. (18) is motivated from our assumption that the manifold is locally circular (with curvature $\sqrt{c_i}$) in each of the principal directions. r_i is determined so that the perpendicular distance from the manifold becomes the prescribed distance $E_{\mathcal{M}}$ for the point at the distance r_i from the root node in the principal direction (see Fig. 3). By the above rules with the choices of $\rho_{\min}$ and $\rho_{\max}$ as in (19), we limit the range of r_i; the minimum value of r_i is assured of being no smaller than the basic stepsize in the RRT extension step, while the maximum value of r_i cannot exceed the distance between the initial and final nodes. Note that the basis and bound need to be computed only once for each root node. Once the principal basis and the bounds are determined as above, random sampling on the tangent space can be done as in (16).

3.5. Creating a new tangent space

When the distance from the extended node q_{new} to $\mathcal{M}$ exceeds a certain threshold, denoted $E_{\mathcal{M}}$, the tangent space no longer approximates $\mathcal{M}$ with the desired accuracy. In such cases we project the extended node onto $\mathcal{M}$ using our previously described projection algorithm 6, and treat the projected point as a new root node for a new tangent space. For each tangent space generated we store the index of the root node, the projection matrix P, the principal basis, and the bounds as evaluated in (17) and (18).

When a new tangent space is created, it is important to ensure that the new tangent space not overlap significantly with the same area of $\mathcal{M}$ covered by the parent tangent space. To prevent such overlapping, we adopt the following two procedures.

3.5.1. Preventing backtracking on tangent spaces. We compute the vector from the root node of the parent tangent space to the projected node that will be the new root node; let the projection of this vector onto the new tangent space be denoted by d_1. In the random sampling phase, we only use a tangent half-space to ensure that we do not "backtrack" to already explored regions of $\mathcal{M}$; when the inner product of the random vector on the tangent space $\sum_{i=1}^{m} w_i b_i$ in (16) and d_1 is negative, we invert the direction of $\sum_{i=1}^{m} w_i b_i$.

3.5.2. Preventing overlapping tangent spaces. If the root nodes of two tangent spaces are close to each other, the tangent spaces will then tend to overlap each other. To reduce the probability of this happening, we discard random samples if their nearest neighbor nodes have been projected (and declared to be a root node of a new tangent space) or are parent nodes of a projected node.

3.6. Dynamic domain sizing of tangent space

As pointed out in Yershova *et al.*,[19] balancing the amount of refinement and expansion is particularly important in determining the performance of RRT algorithms. Our approach is similar to the algorithm introduced in ref. [19] in the way that (i) the searching domain is incrementally enlarged as the trees are expanded to unexplored areas of the configuration space and (ii) the balancing between the

exploration and the refinement is controlled by a certain parameter—in ref. [19] the dynamic domain radius determines the balance, while in our algorithm the size of the tangent space domain does. In our algorithm, the size of the tangent space domain is determined from the prescribed error threshold $E_{\mathcal{M}}$ and local curvature information as derived in (18). It is entirely possible that the constraint manifold may deviate considerably from what is predicted by local curvature. To be probabilistically complete, the algorithm must ensure that new tangent spaces are created to cover unexplored regions of the constraint manifold.

To address these and other issues, we adopt the method of Jaillet *et al.*[6] on adaptively tuning the sampling domain. If the tangent space domain is not large enough for the branches of the trees to spread out to unexplored regions—more specifically, if further extension to the exterior regions cannot be made on the tangent space, or creating a new tangent space is not possible—the tangent space domain should then be enlarged. If on the other hand the tangent space domain is excessively large for the purposes of approximating the constraint manifold, the domain should then be made smaller. The tangent space domain can be modified during the iteration of the algorithm by appropriately balancing these two opposing features. Algorithm 9 provides an example of our dynamic domain function by adopting the following rules:

- If the new extended node in the exterior of the tangent space is not projected, the tangent space domain is enlarged by a certain ratio (1.2 for the purposes of this paper).
- If the new extended node near the root node of the tangent space needs to be projected, the tangent space domain is decreased by a certain ratio (0.8 for the purposes of this paper).

Algorithm 9 DynamicDomain(q_{new}, TangentSpace)

```
q_new ← q_new, q_root ← TangentSpace.q_root
dist1 ← ‖q_new − q_root‖
dist2 ← ‖TangentSpace.domain‖
if q_new is projected then
    if dist1 > 0.9 ∗ dist2 then
        TangentSpace.domain ← 1.2 ∗ TangentSpace.domain
    end if
else if dist1 < 0.4 ∗ dist2 then
    TangentSpace.domain ← 0.8 ∗ TangentSpace.domain
end if
return TangentSpace
```

3.7. Selection bias for tangent spaces

The trees generated by the TB-RRT algorithm consist of multiple tangent spaces and branch nodes, such that every extended node belongs to one of the two trees (one emanating from the start node, one from the goal node—recall that our algorithm is also bidirectional), and also to one of the tangent spaces. When nodes are randomly sampled, we first need to choose a tangent space among all of the tangent spaces created.

One possible criterion for biasing the random selection of tangent spaces is to use the local curvature information of $\mathcal{M}$. Recall that the greater the extrinsic curvature, the greater the error in approximating the constraint manifold by the tangent space. Tangent spaces attached at points with lower curvature can therefore be favored over those attached to points with higher curvature. In particular, if one wants to uniformly sample over the entire tangent space, the absolute value of the Gaussian curvature, given by the determinant of the second fundamental form, can be used as a bias factor since the area of the sampling domain is set to be proportional to this value.

Since the entire sampling domain grows as new tangent spaces are created during the iteration, one can add another heuristic bias to favor newly discovered areas (new tangent spaces) over an area that has been sufficiently explored. One means of ensuring that recently created tangent spaces are selected more often is to use the number of nodes that belong to the tangent space as a selection criterion. That is, if a certain tangent space has fewer nodes than the others, we give it a greater chance

of being selected. This heuristic can help balance the distribution of the RRT nodes over the entire collection of tangent spaces.

3.8. Connection test

As mentioned earlier, the TB-RRT algorithm is, like,[1] bidirectional in the sense that two trees—one each emanating from the initial and goal nodes—are simultaneously generated. When the algorithm tries to connect between the extended node and the nearest node on the opposite tree, the inner products are first evaluated between the connecting line vector and the rows of $J(q)$ (recall that the rows of $J(q)$ are orthogonal to the surface); if the inner products are sufficiently close to zero at both ends, the connecting segment is deemed to be sufficiently close to the tangent spaces. In this case uniformly spaced points along the connecting segment are tested to see if the inequality constraints are satisfied. If at any of these points the constraints are violated, or the error $\|e\|$ exceeds the prescribed threshold, no connection is established.

4. A Simple Variation of TB-RRT

We now present a simple variation of the TB-RRT algorithm, in which the random sampling is performed in the ambient configuration space instead of on the tangent spaces to the constraint manifold. As before assume that the given q_{init} and q_{final} are already on the configuration space $\mathcal{M}$, and that two tangent spaces to $\mathcal{M}$ have been constructed at q_{init} and q_{final}. Starting with q_{init} and q_{final} as the root nodes, two trees are then grown on the tangent spaces via the TB-RRT algorithm shown in Algorithm 10:

1. A random sample node q_{rand} is first generated on the configuration space. Given q_{rand}, we find the nearest neighbor node q_{near} from the first tree with q_{init} as its root node. We then iteratively extend the tree a fixed distance from q_{near} in the direction of q_{rand}. This simplified version can also be classified as RRT-ExtCon or RRT-ConCon. The two different extension phase algorithms are described in Algorithms 11 and 12.
2. For each iteration of the node extension, the new node generated is projected onto the tangent space on which the nearest neighbor node lies. The extended node q_{new} is then tested to see if the inequality constraints are satisfied.
3. For every extended node q_{new}, we evaluate the distance from q_{new} to the constraint manifold $\mathcal{M}$. If the distance exceeds a certain threshold, the node is projected onto $\mathcal{M}$, and a new tangent space is created at the projected node (see Fig. 4). In the simplified TB-RRT algorithm only the index of each tangent space root node and the projection matrix $P(q_{root}) \in \mathbb{R}^{n\times n}$ are evaluated and stored; the tangent space basis is no longer necessary. The extension of the tree is continued until the extended node fails to satisfy the inequality conditions, or the tree can no longer be extended toward q_{rand}.
4. If the extension from the first tree ends, an attempt is made to connect q_{new} with the opposite tree. If successful, a final path has been found and the algorithm terminates. Otherwise, the nearest neighbor node on the other tree is found to the the final extended node of the first tree. The same extension algorithm is applied to the other tree, and another attempt is made to connect q_{new} with the first tree.
5. The algorithm repeats the above process with swapping the trees until it successfully terminates or exceeds the maximum number of iterations.
6. After the final path is determined through the collection of tangent bundles, the nodes on the final path are projected onto the constraint manifold (see Algorithm 8).

5. Case Studies

We provide three sets of numerical experiments using both the main and simplified versions of the TB-RRT algorithm. The case studies are evaluated on a PC with a 2.93GHz Inter(R) core(TM) i7 quad processor and 4.0GB RAM. The MPNNN algorithm of Yershova *et al.*[18] is used to find the nearest neighbor nodes. Our results are compared with the CBiRRT algorithm.[1] In terms of the classification defined by LaValle and Kuffner *et al.*,[9] the CBiRRT algorithm as originally formulated is classified as

Algorithm 10 TB-RRT-Simple(q_{init}, q_{final})

```
Tree_a, Tree_b, TangentSpaces[]
Tree_a.AddVertex(q_init), Tree_b.AddVertex(q_final)
TangentSpaces.Add(CreateTangentSpace(q_init))
TangentSpaces.Add(CreateTangentSpace(q_final))
while i < I_max do
    i ← i + 1
    q_rand ← RandomSampling()
    q_near ← FindNearestNodeOnTree(Tree_a, q_rand)
    if q_new ← ExtendSimple(Tree_a, q_near, q_rand) then
        if Connect(q_new, Tree_b) then
            Path ← ExtractPath()
            return LazyProjection(Path)
        end if
    else
        q_new ← q_near
    end if
    q_near ← FindNearestNodeOnTree(Tree_b, q_new)
    if q_new ← ExtendSimple(Tree_b, q_near, q_new) then
        if Connect(q_new, Tree_a) then
            Path ← ExtractPath()
            return LazyProjection(Path)
        end if
    end if
    Swap(Tree_a, Tree_b)
end while
return NULL
```

Algorithm 11 ExtendSimple(Tree, q_{near}, q_{target}) *(RRT-ExtCon)*

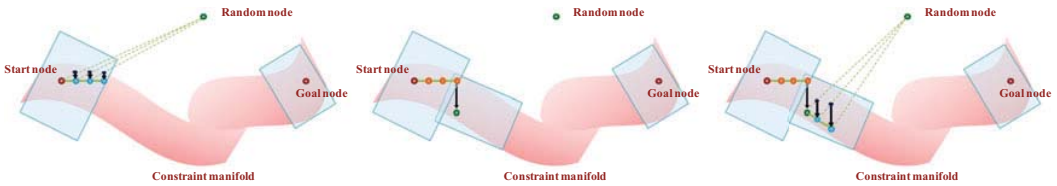
```
q_new ← ExtendOneStep(q_near, q_target)
q_new ← ProjectOntoTangentSpace(q_new, q_near)
if CollisionOccur(q_new) then
    return NULL
end if
if ‖f(q_new)‖ > E_M then
    if ProjectionNewtonRaphson(q_new) then
        TangentSpaces.Add(CreateTangentSpace(q_new))
    else
        return NULL
    end if
end if
Tree.AddVertex(q_new), Tree.AddEdge(q_old, q_new)
return q_new
```

RRT-ConCon. For comparison purposes, we implement both RRT-ExtCon and RRT-ConCon modes for the CBi-RRT algorithm.

A parameter that critically affects the performance of any RRT algorithm is the stepsize: an excessively large stepsize may cause low motion resolution and eventually failure, while an excessively small stepsize consumes a considerable amount of computing resource in the extension phase. Ideally the choice of stepsize should take into account the volume of the configuration space while achieving the desired level of motion resolution. For the general class of problems considered in this paper a stepsize on the order of $10^{-2} \sim 10^{-3}$ seems to be appropriate, and for our subsequent experiments the stepsize is fixed to 0.05. The ε appearing in Algorithm 6 denotes the allowed

Algorithm 12 ExtendSimple(Tree, q_{near}, q_{target}) *(RRT-ConCon)*

$q_{\text{new}} \leftarrow$ NULL, $q_{\text{old}} \leftarrow q_{\text{near}}$
while (true) **do**
  $q_r \leftarrow$ ExtendOneStep(q_{old}, q_{target})
  $q_r \leftarrow$ ProjectOntoTangentSpace(q_r, q_{old})
  if CollisionOccur(q_r) **then**
    return q_{new}
  end if
  if $\|f(q_r)\| > E_{\mathcal{M}}$ **then**
    if ProjectionNewtonRaphson(q_r) **then**
      TangentSpaces.Add(CreateTangentSpace(q_r))
    else
      return q_{new}
    end if
  end if
  if $\|q_{\text{target}} - q_r\| > \|q_{\text{target}} - q_{\text{old}}\|$ **then**
    return q_{new}
  end if
  $q_{\text{new}} \leftarrow q_r$
  Tree.AddVertex(q_{new}), Tree.AddEdge(q_{old}, q_{new})
  $q_{\text{old}} \leftarrow q_{\text{new}}$
end while
return q_{new}



(a) Extend the tree on the nearest tangent space as far as possible toward q_{rand}.

(b) Project the node and generate a new tangent space when the distance $\|e\|$ from the extended node q_{new} to $\mathcal{M}$ exceeds $E_{\mathcal{M}}$.

(c) The extension of the branch can be stopped in the event of collisions or constraint violations.

Fig. 4. Illustration of the Simplified TB-RRT algorithm.

numerical error for projecting a node onto the manifold; for our subsequent experiments, ε is fixed to 10^{-5} for all the algorithms.

We also introduce another user-specified parameter that influences algorithm performance, the error constraint threshold $E_{\mathcal{M}}$. This parameter is used for the purposes of comparing with the norm of the constraint error, and we can deduce a range for this parameter via a physical interpretation of the constraint equations. For our experiments we use varying values of $E_{\mathcal{M}}$ for comparison purposes. In each of our case studies, the initial and final configurations of the robots are given, and all experimental results are reported as the average of 100 trials performed with different random seeds.

5.1. Planning on a torus

We first consider a planning problem on a two-dimensional torus embedded in three-dimensional ambient space (see Fig. 5). The one-dimensional constraint equation is simply $f(x_1, x_2, x_3) = (R - \sqrt{x_1^2 + x_2^2})^2 + x_3^2 - r^2 = 0$, where the radii R and r are set to 1.0 and 0.5, respectively. Given the constraint equation, the lowest value of $f(x_1, x_2, x_3)$ is -0.25 when the configurations lie on the inner circle of $x_1^2 + x_2^2 + x_3^2 = 1.0$. From this we can deduce that $E_{\mathcal{M}}$ needs to be specified lower than 0.25. We use three sets of $E_{\mathcal{M}}$ values (0.1, 0.15 and 0.2) for each of the experiments (see Table I).

Table I. Torus problem.

	TB-RRT (full version)						TB-RRT (simple version)						CBiRRT	
	ConCon			ExtCon			ConCon			ExtCon			ConCon	ExtCon
$E_{\mathcal{M}}$	0.1	0.15	0.2	0.1	0.15	0.2	0.1	0.15	0.2	0.1	0.15	0.2	·	·
Iteration	486	414	275	4551	2707	1321	259	216	268	748	1038	796	374	1486
Total nodes	1802	1732	1319	4308	2670	1388	571	546	588	870	1181	923	17867	1532
Tangent spaces	91	72	49	122	59	27	40	31	24	322	501	316	·	·
Final path length	133	139	136	118	114	112	115	117	118	116	117	115	573	165
Time (msec)	23	21	15	80	43	20	5	4	5	9	13	10	110	28

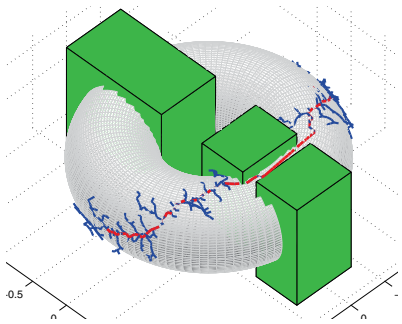


Fig. 5. In the torus problem, random trees are extended on the tangent spaces of the torus: the start and goal configuration coordinates are given by $(1.5, 0.0, 0.0)$ and $(-1.5, 0.0, 0.0)$, respectively, and the green boxes impose inequality constraints on the feasible configuration space.

For the TB-RRT algorithm, the number of projections to the constraint manifold is approximately equal to the sum of the tangent spaces and the nodes in the final path, whereas in the CBiRRT algorithm, it is approximately equal to the total number of nodes. In the RRT-ConCon mode experiments, the full version of the TB-RRT algorithm consumes less time than the CBiRRT algorithm. Also, as $E_{\mathcal{M}}$ is increased, fewer projections occur, reducing the overall computation time. When $E_{\mathcal{M}}$ is set to be 0.1 and 0.15, even though the full TB-RRT algorithm involves considerably fewer projections to the constraint manifold, the overall computation time is greater. We speculate that the likely reason can be traced to the curvature computations, which for the torus problem offer little if any benefit for the additional computational effort.

However, when $E_{\mathcal{M}}$ is set to be 0.2 (which is still lower than the allowed margin), the computational performance of the full version of the TB-RRT algorithm is considerably better than that of the CBiRRT algorithm. For this structurally simple problem, the most basic version of the TB-RRT algorithm consumes less time overall than the other algorithms. We also find that the error threshold parameter affects the number of tangent spaces created during runtime execution of the algorithm, but that overall it has minimal effect on the algorithm's total computation time for the simple version of the TB-RRT algorithm. For this rather simple constraint manifold, random sampling on tangent spaces does not provide much advantage in exploring the constraint manifold over random sampling on the ambient configuration space. The advantages of curvature-based local exploration are greatest for complex manifolds with nonuniform curvatures.

To evaluate the effect of the strategies introduced in Sections 3.5.1 and 3.5.2, we perform experiments with the same setup, but without applying these strategies in the full version of the TB-RRT algorithm (see Table II). The number of iterations, total nodes and tangent spaces are all significantly increased when the strategies are not applied, as are the computational times.

5.2. Planar eight-bar linkage

In the next example we consider a planar eight-bar linkage with a single loop constraint as illustrated in Fig. 6. The tool tip is attached to the fourth link of the system. The loop closure constraint equation

Table II. Torus Problem (without the strategies).

	TB-RRT (full version)					
	ConCon			ExtCon		
$E_{\mathcal{M}}$	0.1	0.15	0.2	0.1	0.15	0.2
Iteration	702	646	526	6231	5953	9496
Total nodes	3245	3530	3135	6138	4512	6447
Tangent spaces	194	178	138	315	203	229
Final path length	141	156	156	121	117	117
Time (msec)	51	49	39	147	131	161

(a) Initial configuration (b) Final configuration (c) Trajectory determined by the TB-RRT algorithm

Fig. 6. Planar eight-bar linkage: the object grasped by the end-effector is trapped by the side obstacles in the initial and final configurations.

is of the form

$$f(q) = \begin{pmatrix} \theta_e \\ x_e \\ y_e \end{pmatrix} \tag{20}$$

where θ_e denotes the orientation of the frame of the last link and (x_e, y_e) the Cartesian position, $q \in \mathbb{R}^8$ denotes the joint coordinates, and $f : \mathbb{R}^8 \to \mathbb{R}^3$ is the forward kinematics mapping of the last link. The planar eight-bar linkage has a configuration space that can be regarded as a five-dimensional surface embedded in $\mathbb{R}^8$ (more accurately, a three- dimensional surface embedded in an eight-dimensional flat torus T^8). In this example, each of the link lengths are set to $0.3\,m$, and the end-effector is assumed to grasp a rectangle of size $0.15\,m \times 0.4\,m$ (see Fig. 6). Based on the workspace size and the nature of the constraint equations, we use three sets of $E_{\mathcal{M}}$ values (0.01, 0.03 and 0.05) for each experiment (see Table III). The box-shaped obstacles impose inequality constraints on the feasible configuration space. In the initial and final configurations of this example, the manipulators are surrounded by obstacles as shown in Figs. 6(a) and 6(b). Such a placement of obstacles causes the configuration space to have the shape of a "bug trap". In such cases, the size of the sampling domain becomes more critical to algorithm performance.

For this example, the full version of TB-RRT takes considerably less time overall than both the CBiRRT or the simplified version of TB-RRT (with the exception of the RRT-ConCon case for $E_{\mathcal{M}} = 0.01$, see Table III).

5.3. Two cooperating KUKA arms

In the third example we consider two six-DOF KUKA manipulators assigned with the task of moving a cylindrical bar in three-dimensional space. The manipulator's end-effectors are assumed to rigidly hold the opposite ends of the cylindrical bar. Each robot is about 2 meters in height, and the workspace size is also approximately one cubic meter (see Fig. 7). The dimension of the configuration space is twelve $(6 + 6)$, and subject to a six-dimensional kinematic loop constraint. The kinematic closure

Table III. Planar 8-bar linkage.

	TB-RRT (full version)						TB-RRT (simple version)					
	ConCon			ExtCon			ConCon			ExtCon		
$E_{\mathcal{M}}$	0.01	0.03	0.05	0.01	0.03	0.05	0.01	0.03	0.05	0.01	0.03	0.05
Iteration	5123	1815	2066	34995	7780	6042	1779	2534	3871	9944	10813	14601
Total nodes	4149	3130	3146	6503	2209	1650	5328	6204	7573	2070	1698	2276
Tangent spaces	280	126	94	287	47	22	476	276	215	130	57	66
Final path length	274	275	300	255	234	234	418	365	372	288	265	285
Time (msec)	397	210	269	1558	396	394	224	306	469	509	550	843

	CBiRRT	
	ConCon	ExtCon
Iteration	1918	11188
Total nodes	13461	2708
Final path length	1100	466
Time (msec)	1581	2727

equations for this problem can be expressed as

$$h_1(q_1) = h_2(q_2), \tag{21}$$

where $h_i : \mathbb{R}^6 \rightarrow SE(3)$, $i = 1, 2$ are the forward kinematics maps of the two KUKA manipulators, and $q_1, q_2 \in \mathbb{R}^6$ denotes their respective joint coordinates.

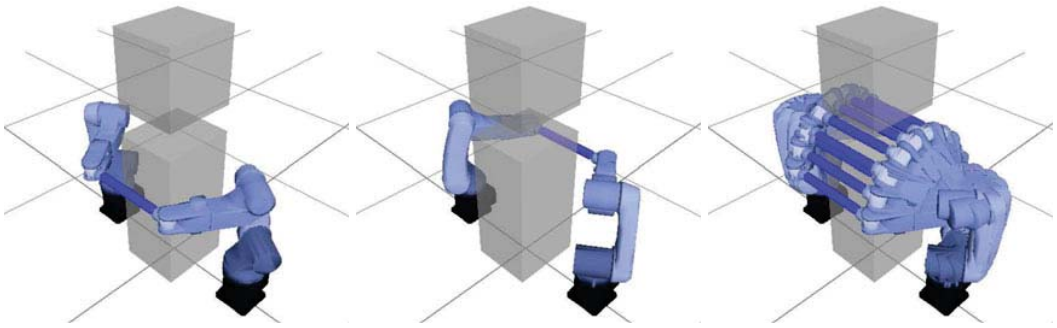
The closed-form constraint equations $F : \mathbb{R}^{12} \rightarrow SE(3)$ can be defined as

$$F(q) = h_1(q_1)^{-1} h_2(q_2) = I, \tag{22}$$

where $q = (q_1, q_2)$ denotes the coordinates of the total configuration space. The matrix form of the left differentiation $F^{-1} \frac{\partial F(q)}{\partial q}$ can then be considered as the constraint Jacobian defined in Eq. (4). The analytic form of Eqs. (7) and (8) for this constraint equation can be found in ref. [25].

For our case studies we use the set of $E_{\mathcal{M}}$ values 0.1, 0.15, and 0.2 for each of the experiments (see Tables IV and V). The performance of the simple and the full versions of the TB-RRT algorithm are not that different from the perspective of computational time, but do differ in the number of tangent spaces and nodes created. Compared to the RRT-ConCon algorithm, the CBiRRT algorithm requires fewer iterations and number of nodes than either version of TB-RRT, but still takes more time. The likely reason is that the TB-RRT algorithms project only the root nodes of the tangent spaces and the nodes in the final path, whereas the CBiRRT algorithm projects every extended node onto the constraint manifold.

We now compare the performance of the TB-RRT algorithm with the AtlasRRT algorithm. In ref. [24] the example of two cooperating manipulator arms is given, but without any obstacles in the environment, and for comparison purposes we also apply the TB-RRT algorithm under the same set of conditions (see Table V). Applying the AtlasRRT algorithm, the number of charts (20) and



(a) Initial configuration (b) Final configuration (c) Trajectory determined by the TB-RRT algorithm

Fig. 7. Two cooperating KUKA manipulators: the transparent gray boxes are workspace obstacles. The two KUKA arms must hold the cylinder bar together while avoiding the obstacles.

Table IV. Two Cooperating KUKA manipulators.

	TB-RRT (full version)						TB-RRT (simple version)					
	ConCon			ExtCon			ConCon			ExtCon		
$E_{\mathcal{M}}$	0.1	0.15	0.2	0.1	0.15	0.2	0.1	0.15	0.2	0.1	0.15	0.2
Iteration	395	566	985	1144	945	1043	490	519	268	808	835	972
Total nodes	3495	3917	5304	1195	1022	1061	948	910	1035	732	761	779
Tangent spaces	60	52	57	15	12	11	38	25	22	49	70	84
Final path length	411	415	429	294	280	284	313	297	290	239	241	239
Time (msec)	199	231	324	144	115	122	160	173	260	127	114	158

	CBiRRT	
	ConCon	ExtCon
Iteration	269	1033
Total nodes	1310	985
Final path length	368	399
Time (msec)	386	607

the total number of nodes in the final path (366) is of the same order as for the TB-RRT algorithm (approximately 20∼30 tangent spaces and 300∼500 nodes depending on the particular version of TB-RRT). These numbers lead us to conjecture that the resulting paths obtained by TB-RRT and AtlasRRT should in most cases be qualitatively similar. An examination of the computation times is illuminating, however. Although a precise comparison is not possible due to differences in implementation and hardware (although any differences in hardware performance specifications seem to be minor), in terms of computational times the TB-RRT algorithm is more efficient by nearly three orders of magnitude: whereas it takes 14.24 seconds with the AtlasRRT algorithm, the TB-RRT algorithm consumes only 70 milliseconds in the worst case. The need to compute volume intersections in the

Table V. Two Cooperating KUKA manipulators (without obstacles).

	TB-RRT (full version)						TB-RRT (simple version)					
	ConCon			ExtCon			ConCon			ExtCon		
$E_{\mathcal{M}}$	0.1	0.15	0.2	0.1	0.15	0.2	0.1	0.15	0.2	0.1	0.15	0.2
Iteration	29	23	19	247	229	226	8	7	8	164	138	123
Total nodes	950	819	706	529	499	496	405	374	236	363	320	297
Tangent spaces	28	21	16	15	11	9	21	15	11	25	24	25
Final path length	304	301	296	218	218	217	255	250	236	220	211	204
Time (msec)	70	65	62	51	51	53	46	46	44	46	46	47

AtlasRRT algorithm likely accounts for the large discrepancy in computation times with the TB-RRT algorithm.

6. Conclusion

This paper has presented a new randomized algorithm for task constrained motion planning, the TB-RRT algorithm (and a simpler variant). Unlike existing RRT algorithms for constrained motion planning, the distinguishing feature of the TB-RRT algorithm is that RRTs are constructed in the tangent spaces of the constraint manifold, and projected to the manifold only when the newly sampled nodes exceed a certain threshold distance. Our method can be contrasted with existing approaches that randomly sample in the configuration space, and project every sample back to the constraint manifold; such a procedure can result in nodes that only extend the tree minimally, possibly revisiting previously explored directions.

Extensive numerical studies with our algorithm suggest that even with the increased complexity and bookkeeping, it is in general more reliable and computationally efficient than existing algorithms. The performance advantages of TB-RRT appear to increase with the complexity of the planning problem (e.g., increasing problem dimension number of constraints and obstacles). Using local curvature information to bound the tangent space domains also seems to be quite helpful in the case studies we have examined.

The algorithm relies on a number of heuristic techniques in, e.g., selecting tangent spaces, preventing backtracking, and creating tangent spaces that are spaced sufficiently apart, and these techniques can be refined depending on the features of the motion planning problem. More effective methods for exploiting local curvature information about the constraint manifold, as well as finding ways to include, e.g., dynamics, nonholonomic constraints, and other physical constraints into the planning framework, are the subject of future work.

In this paper our focus has been on the case when the configuration space is defined by a single constraint manifold of fixed dimension m. It is relatively straightforward to extend to the case in which the configuration space consists of several constraint manifolds of different dimension. Each time a new node is created, the constraint equations are checked; if the dimension of the equality constraint (m) is changed, the node is projected back to the constraint manifold, and a new tangent space of dimension $n - m$ is created. This node point can be considered to be on the intersection of the two different manifolds. If the equality constraint vanishes ($m = 0$), then this implies a full-dimensional manifold, i.e., the constraint manifold now becomes a volume within the ambient Euclidean space, and in this case the domain size is set to the prescribed maximum value.

Finally, in certain constrained planning problems for kinematically redundant robots, a task space trajectory is sometimes specified *a priori*. Berenson *et al.*[2] develop an extension of their CBiRRT algorithm in which the task space trajectory is segmented by segmented points and an RRT constructed

on the foliation. We expect that TB-RRT can also be extended in this way to address the planning of constrained motions when task space trajectories are prescribed.

Acknowledgements
Financial support for this research was provided in part by Center for Intelligent Robotics, Center for Advanced Intelligent Manipulation, Biomimetic Robotics Research Center, SNU-BK21+ Program in Mechanical Engineering, and SNU-IAMD.

References

1. D. Berenson, S. Srinivasa, D. Ferguson and J. Kuffner, "Manipulation Planning on Constraint Manifolds," *Proceedings of the IEEE International Conference Robotics and Automation* (2009) pp. 625–632.
2. D. Berenson, S. Srinivasa and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *Int. J. Robot. Res.* **30**(12), 1435–1460 (2011).
3. J. Cortés and T. Siméon, "Probabilistic Motion Planning for Parallel Mechanisms," *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 3 (2003) pp. 4354–4359.
4. L. Han, L. Rudolph, J. Blumenthal and I. Valodzin, "Stratified Deformation Space and Path Planning for a Planar Closed Chain with Revolute Joints," **In**: *Algorithmic Foundation of Robotics VII (WAFR 2006), Springer Tracts in Advanced Robotics*, vol. 47 (S. Akella, N. Amato, W. Huang and B. Mishra, eds.) (Springer New York, 2006) pp. 235–250.
5. M. E. Henderson, "Multiple parameter continuation: Computing implicitly defined k-manifolds," *Int. J. Bifurcation Chaos* **12**(3), 451–476 (2001).
6. L. Jaillet, A. Yershova, S. LaValle and T. Simeon, "Adaptive Tuning of the Sampling Domain for Dynamic-Domain RRTs," *Proceedings of the IEEE/RSJ International Conference Intelligent Robots and Systems* (2005) pp. 2851–2856.
7. Y. Koga, K. Kondo, J. Kuffner and J. Latombe, "Planning Motions with Intentions," *Proceedings of the Siggraph* (1994) pp. 395–408.
8. S. LaValle, Rapidly-Exploring Random Trees: A New Tool for Path Planning, TR 98-11 (Computer Science Dept., Iowa State University, Oct. 1998).
9. S. LaValle and J. Kuffner, "Rapidly Exploring Random Trees: Progress and Prospects," **In**: *Algorithmic and Computational Robotics: New Directions* (B. R. Donald, K. M. Lynch and D. Rus, eds.) (A K Peters, Wellesley, MA, 2001) pp. 293–308.
10. G. Sanchez and J. Latombe, "A Single Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking," **In**: *Springer Tracts in Advanced Robotics*, vol. 6 (Springer, 2003) pp. 403–417.
11. S. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," *Int. J. Humanoid Robot.* **2**(4), 505–518 (2005).
12. N. Shvalb, L. G. Liu, M. Shoham and J. C. Trinkle, "Motion planning for a class of planar closed-chain manipulators," *Int. J. Robot. Res.* **26**(5), 457–474 (2007).
13. M. Stilman, "Task Constrained Motion Planning in Robot Joint Space," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2007) pp. 3074–3081.
14. X. Tang, S. Thomas, P. Coleman and N. Amato, "Reachable distance space: efficient sampling-based planning for spatially constrained systems," *Int. J. Robot. Res.* **29**(7), 916–934 (2010).
15. T. T. Um, C. Suh, B. Kim and F. C. Park, "Tangent Space RRT with Lazy Projection: An Efficient Planning Algorithm for Constrained Motions," **In**: *Advances in Robot Kinematics* (J. Lenarcic and M. Stanisic, eds.) (Springer Netherlands, 2010) pp. 251–260.
16. J. H. Yakey, S. LaValle and L. Kavraki, "Randomized path planning for linkages with closed kinematic chains," *IEEE Trans. Robot. Autom.* **17**(6), 951–958 (2001).
17. K. Yamane, J. Kuffner and J. Hodgins, "Synthesizing animations of human manipulation tasks," *ACM Trans. Graph.* **23**(3), 532–539 (2004).
18. A. Yershova, MPNN: Nearest Neighbor Library for Motion Planning, (2005) [Online]. Available at http://msl.cs.uiuc.edu/ yershova/MPNN/MPNN.htm.
19. A. Yershova, L. Jaillet, T. Simeon and S. LaValle, "Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain," *Proceedings of the IEEE International Conference on Robotics and Automation* (2005) pp. 3856–3861.
20. S. Lindermann and S. LaValle, "Incrementally Reducing Dispersion by Increasing Voronoi Bias in RRTs," *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4 (2004) pp. 3251–3257.
21. C. Suh, T. Um, B. Kim, H. Noh, M. Kim and F. C. Park, "Tangent Space RRT: A Randomized Planning Algorithm on Constraint Manifolds," *Proceedings of the IEEE International Conference on Robotics and Automation* (2011) pp. 4968–4973.
22. C. Suh, B. Kim and F. C. Park, "The Tangent Bundle RRT Algorithms for Constrained Motion Planning," *Proceedings of the 13th World Congress in Mechanism and Machine Science*, Guanajuato, Mexico (2011).

23. J. M. Porta and L. Jaillet, "Path Planning on Manifolds Using Randomized Higher-Dimensional Continuation," **In**: *Algorithmic Foundations of Robotics IX (WAFR 2010)* (D. Hsu, V. Isler, J.C. Latombe and M. Lin, eds.) (Springer Berlin Heidelberg, 2011) pp. 337–353.
24. L. Jaillet and J. M. Porta, "Path Planning with Loop Closure Constraints Using an Atlas-Based RRT," *Proceedings of the 15th International Symposium on Robotics Research (ISRR)*, Flagstaff, USA (2011).
25. F. C. Park and J. Bobrow, "Geometric optimization algorithms for robot kinematic design," *J. Robot. Syst.* **12**(6), 453–463 (1995).