Advanced Robotics

ISSN: 0169-1864 (Print) 1568-5535 (Online) Journal homepage: https://www.tandfonline.com/loi/tadr20

Randomized path planning for tasks requiring the release and regrasp of objects

Jinkyu Kim, Inyoung Ko & Frank C. Park

To cite this article: Jinkyu Kim, Inyoung Ko & Frank C. Park (2016) Randomized path planning for tasks requiring the release and regrasp of objects, Advanced Robotics, 30:4, 270-283, DOI: 10.1080/01691864.2015.1105150

To link to this article: https://doi.org/10.1080/01691864.2015.1105150

ADVANCED ROBOTICS, 2016
VOL. 30, NO. 4, 270–283
http://dx.doi.org/10.1080/01691864.2015.1105150

FULL PAPER

Randomized path planning for tasks requiring the release and regrasp of objects

Jinkyu Kim[a], Inyoung Ko[b] and Frank C. Park[a]

[a]Department of Mechanical and Aerospace Engineering, Seoul National University, Seoul 151-742, Korea; [b]Global Technology Center, Samsung Electronics Co., Ltd, Suwon 443-742, Korea

ABSTRACT

This paper presents a randomized planning algorithm for manipulation tasks that require the robot to release and regrasp an object in different robot postures. Such problems arise, for example, in robotic suturing and knot tying, and in assembly tasks where parts must be guided through complex environments. Formulating the problem as one of planning on a foliated manifold, we present a randomized planning algorithm that, unlike existing methods, involves sampling and tree propagation primarily in the task space manifold; such an approach significantly improves computational efficiency by reducing the number of projections to the constraint manifold, without incurring any significant increases in the number of release-regrasp sequences. We also propose a post-processing topological exploration algorithm and path refinement procedure for reducing the number of release-regrasp sequences in a solution path, independent of the algorithm used to generate the path. Experiments involving spatial open chains with up to 10 degrees of freedom, operating in complex obstacle-filled environments, show that our algorithm considerably outperforms existing algorithms in terms of computation time, path length, and the number of release-regrasp operations.

ARTICLE HISTORY
Received 30 March 2015
Revised 6 August 2015
Accepted 26 September 2015

KEYWORDS
Motion planning; sampling-based planning; foliation; release-regrasp; rapidly exploring random tree

1. Introduction

In more traditional robot motion planning problems, one is given the robot's initial and desired final configurations – described by two points in the robot's configuration space manifold – and the objective is to find a curve lying on the feasible region of the configuration space that connects these two given endpoints. However, many robot tasks require the release and regrasp of objects in different arm postures. For example, in robotic suturing, a robot must first grasp a needle by its base and insert it through one side of tissue, release the needle, then regrasp it by its tip and pull it through the tissue. Similar requirements arise in shoe tying and various assembly tasks (see Figure 1). This paper addresses the problem of motion planning for such robotic tasks.

Tasks requiring the release and regrasp of objects are most naturally formulated as a planning problem on a **foliated manifold**, whose structure we discuss in more detail below. The literature on planning problems of this type has thus far been limited. The simpler problem of planning on a curved configuration space–also referred to as a **constraint manifold** since such spaces typically arise out of closed chain and other holonomic task constraints imposed on the robot – without releasing and regrasping of objects has received recent attention, e.g. randomized planning algorithms that exploit rapidly exploring random trees (RRT) have been proposed in [1–3]. In [3] the foliation structure is also introduced for the first time in a redundant manipulator trajectory tracking context.

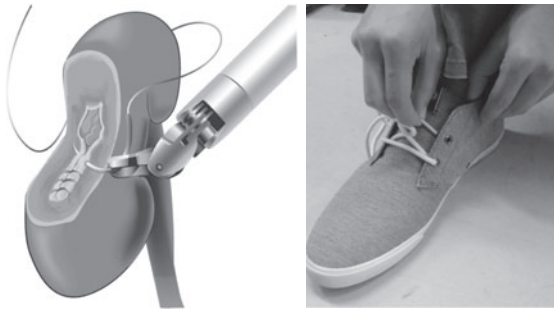

Figure 1. Tasks that require the release and regrasp of objects in different arm postures. (a) Robotic suturing. (b) Shoe tying.

For planning problems that require the release and regrasp of objects, Simeon et al. [4] propose a probabilistic roadmap-based (PRM) planner that searches simultaneously over the configuration space of feasible grasps (CG) and of valid object placements (CP); although it is not explicitly identified as such, this space also possesses a foliated structure. An essential step of their PRM-based planner is the construction of a manipulation graph in the union of the CG and CP spaces that captures the topological structure of the problem. Along an edge of

CONTACT Frank C. Park fcp@snu.ac.kr

the CG (denoted a transfer path) the robot grasps and moves the object, while along an edge of the CP (denoted a transit path) the robot moves alone. Not only is obtaining this topological graph highly time consuming but the PRM algorithm inherits the usual properties associated with PRM-based methods, i.e. increased book-keeping and greater algorithmic complexity, but improved efficiency for tasks that are performed in the same environment (see, e.g. [5]). A dual-arm version of [4] is also presented in [6], that has the added feature of attempting to reduce the number of release-regrasp operations, primarily by attempting to connect paths of the same type within a predefined time. [7] presents another PRM-type dual-arm planner for tasks in which an object can reach the goal configuration only by being handed off from one arm to the other. The algorithm first determines the reachable configurations of the left and right arms, and also of their intersection, before constructing an appropriate graph and PRM-type path planner over the feasible search space.

RRT-type planners have also been proposed for planning problems involving the release-regrasp of objects. Two types of RRT planners are proposed in [8]: one evolving in the configuration space $\mathbb{C}$, and one in the task space $\mathbb{T}$. The $\mathbb{C}$-planner constructs two types of trees in the joint configuration space: those that extend on the foliation, and those that extend in the ambient configuration space, with the latter type of tree intended to connect trees of the former type. In contrast, the $\mathbb{T}$-planner constructs a tree on the object configuration space, and associates with every node of this tree a set of feasible inverse kinematics solutions that are randomly distributed on the foliation. In [8] it is reported that the $\mathbb{T}$-planner outperforms the $\mathbb{C}$-planner, largely because fewer projections to the joint space constraint manifold are required.

Planning problems involving release-regrasp sequences are mathematically much more complex, and the associated planning algorithms more sophisticated and computationally demanding than standard configuration space planning problems. An unstated but underlying premise in previous approaches is that release-regrasp sequences are costly and should be reduced as much possible. Existing algorithms have consequently gone to great lengths to minimize the number of release-regrasp sequences, e.g. by generating trees in both the robot and object configuration spaces, thereby considerably increasing computational costs associated with random sampling and projections to the constraint manifold.

This paper also addresses the problem of RRT-based manipulation planning for tasks that involve the release and regrasp of objects. We assume that an initial and final configurations for the manipulated object are given, together with a function that determines whether a given robot arm configuration results in a collision, and a distance function on both the robot and object configuration spaces. Otherwise we do not assume any more detailed geometric information about the underlying configuration space or obstacles. We first show that such problems are most naturally formulated as a planning problem on a foliated configuration space, or foliated manifold.

Our main contribution is a bi-directional RRT algorithm for generating feasible trajectories that is more computationally efficient than existing methods – in some cases by significant margins, as demonstrated in our experiments. A second contribution that is of independent interest from the first is a procedure based on topological exploration of the feasible configuration space that, given any feasible sequence of motions for achieving the task, reduces the number of release-regrasp sequences.

We achieve this improvement in computational efficiency by devising a method that samples and generates a tree only in the object configuration space, and relies on fast inverse kinematics solutions to propagate a feasible trajectory in the joint configuration space. By reducing the number of samples–and thus the number of projections to the joint space constraint manifold – the computational requirements of planning can be greatly reduced. The main potential drawback with this approach is that by not sampling and generating trees in the joint space constraint manifold (and subsequently searching over a limited space of feasible joint space motions), one risks the possibility of making unnecessary release-regrasp operations. However, by seeking release and regrasp postures of the robot that lie on disconnected regions of the corresponding foliation, the number of release-regrasp operations can be kept to levels similar to the best existing methods, but with a tremendous speed-up in computation times and better quality paths.

The topological exploration algorithm, which is of independent interest and can be used in conjunction with other foliation planners, allows the robot to infer information about the topological structure of the foliated manifold; this is done by randomly sampling and clustering points on the foliated manifold. A path refinement procedure then exploits this topological information and eliminates unnecessary release-regrasp sequences. The topological exploration and path refinement procedures can be added as post-processing procedures to any foliation planning algorithm, and can reduce the number of release-regrasp operations with only small additional computational overhead.

The paper is organized as follows. We first show how our problem can be formulated as a planning problem on a foliated manifold (Section 2), then describe the details of our planner (Section 3) and also our topological

exploration and path refinement algorithms (Section 4). Numerical experiments comparing the performance of our algorithm with existing planners are also described (Section 5). A preliminary version of this work has been presented in [9]; the current work offers a significantly enhanced version of the main algorithm, and also includes the topological exploration and path refinement procedures, with more thorough experimental verification over a wider range of robots and environments.

2. Problem formulation and terminology

We begin with an illustrative example. Consider the planar three-link open chain of Figure 2, which must grasp and move an object along a vertical linear path from top to bottom. Assume the links are all of unit length, and label the joint variables by (q_1, q_2, q_3) and the tip Cartesian coordinates by (x_1, x_2). The forward kinematics is then given by

$$x_1 = \cos q_1 + \cos(q_1 + q_2) + \cos(q_1 + q_2 + q_3)$$
$$x_2 = \sin q_1 + \sin(q_1 + q_2) + \sin(q_1 + q_2 + q_3).$$

Ignoring the orientation of the object–imagine the object to be a circular disk, for example–the configuration space for the robot with the tip placed at the initial object configuration is then obtained by solving the inverse kinematics; the resulting set of inverse kinematic solutions will form a closed path in the joint space (q_1, q_2, q_3).[1] If we repeat this procedure for every point along the vertical line, then the resulting robot configuration space will form a cylindrical surface of the type shown in Figure 2(b). If an obstacle is now placed as shown in Figure 2(c), then the shaded region in the joint configuration space will result in collisions. The final feasible configuration space for the linear path tracking task is shown in Figure 2(d).

The cylindrical surface of Figure 2(b) formed by the union of the closed curves is an example of a **foliated manifold**. Each closed curve corresponding to a point $\mathbf{x}$ on the linear path $\mathbf{X}$ is called a **leaf**, denoted L, and the resulting cylindrical surface, denoted $\mathcal{F}$, is the constraint manifold associated with the linear path $\mathbf{X}$. L and $\mathcal{F}$ satisfy the following properties: (i) $L_{\mathbf{x}} \cap L_{\mathbf{x}'} = \emptyset$ if $\mathbf{x} \neq \mathbf{x}'$; (ii) $\cup_{\mathbf{X}} L_{\mathbf{x}} = \mathcal{F}$.

For the example of Figure 2, the solution path in $\mathcal{F}$ can be partitioned into segments. In the first Segment, the robot moves the object from the initial position to some intermediate point on the linear path. The robot then releases the object, and approaches the object from the other side of the obstacle; this portion corresponds to the second segment. In the third and final segment, the robot regrasps and moves the object to its final destination. We shall refer to the first and third segments of the path as a **connected path**, and the second segment corresponding to the release-regrasp sequence as a **jump path**.

Based on the above example, we now formally define our planning problem. Consider a robot with forward kinematics $f : \mathcal{M} \to \mathcal{W}$, where $\mathcal{M}$ denotes the joint configuration space and $\mathcal{W}$ the object configuration space; in most cases $\mathcal{W}$ can be regarded as the task space of the robot. Denote local coordinates for $\mathcal{M}$ and $\mathcal{W}$ by $\mathbf{q}$ and $\mathbf{x}$, respectively. Let $\mathcal{S} \subseteq \mathcal{W}$ be the region of $\mathcal{W}$ to which the object is constrained to lie in; henceforth, we call $\mathcal{S}$ the **object constraint manifold**. $\mathbf{x}_{\text{initial}}$ and $\mathbf{x}_{\text{final}}$ are the initial and final object configurations on $\mathcal{S}$ that are supplied by the user. Ignoring any obstacles in $\mathcal{W}$, for each point $\mathbf{x} \in \mathcal{S}$ the set of all inverse kinematics solutions $f^{-1}(\mathbf{x})$ then forms a leaf, denoted Leaf($\mathbf{x}$). The union of all leaves then constitutes the foliation $\mathcal{F}$ in the joint configuration space $\mathcal{M}$.

The presence of obstacles in $\mathcal{W}$ now renders some regions of $\mathcal{F}$ infeasible. The objective is to find path segments that alternate between connected paths $Q_c \in \mathcal{F}$ and jump paths $Q_j \in \mathcal{M}$ while avoiding collisions. Denoting the total path by the set Q, we have

$$Q = \left\{ Q_c^1, Q_j^1, \ldots, Q_c^N, Q_j^N, Q_c^{N+1} \right\},$$

where the final point of each connected path segment must be the first point of the subsequent jump segment, and the final point of each jump segment must be the first point of the subsequent connected segment. Summarizing, the final path Q must satisfy the following requirements:

(1) $Q_c^n \in \mathcal{F}\ \forall n$ and $Q_j^m \in \mathcal{M}\ \forall m$
(2) $f(Q_c^1(\text{initial point})) = \mathbf{x}_{\text{initial}}, f(Q_c^{N+1}(\text{final point})) = \mathbf{x}_{\text{final}}$
(3) $f(Q_c^n) \in \mathcal{S}\ \forall n$
(4) Q_j^i connects Q_c^i(final point) with Q_c^{i+1}(initial point)
(5) No collisions with obstacles

3. Planning algorithm

Figure 3 illustrates the basic idea behind our planner as described in Algorithm 1. In most sampling-based planning algorithms, a tree consisting of nodes and edges is constructed in the joint configuration space $\mathcal{M}$. For our planner we construct a tree $T_{\mathcal{S}}$ in the object constraint manifold $\mathcal{S}$. Each node $\mathbf{x}$ in $T_{\mathcal{S}}$ has associated with it two containers, CurrConfig and Path. CurrConfig holds a joint space configuration (called by the function ExploreTree()), while Path holds a path in $\mathcal{M}$ connecting the CurrConfig values of $\mathbf{x}$ and its parent node. Each node $\mathbf{x}$ in $T_{\mathcal{S}}$ also has associated with it a leaf,

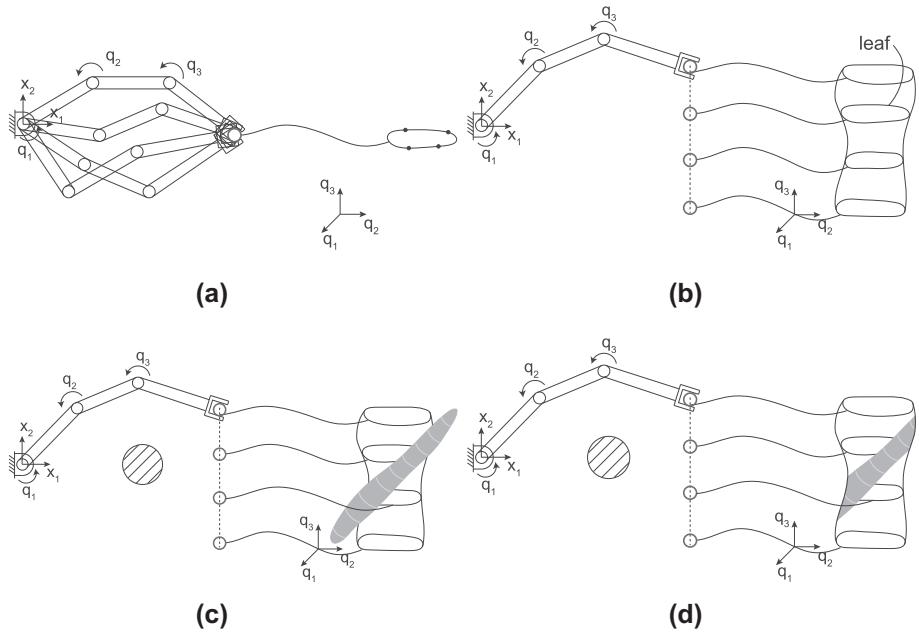

Figure 2. 3R planar open chain example. (a) Multiple inverse kinematics solutions. (b) Object path and corresponding foliation $\mathcal{F}$. (c) With obstacle. (d) Feasible configuration space.

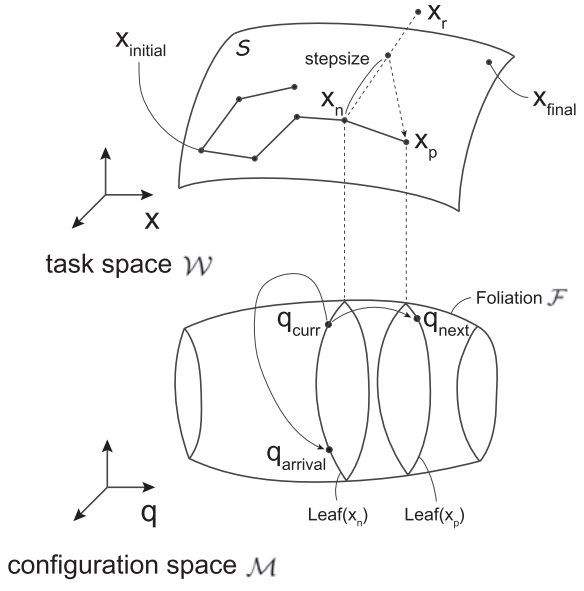

Figure 3. Graphical illustration of the planner.

denoted Leaf($\mathbf{x}$), in the foliated joint space manifold $\mathcal{F}$. All inverse kinematics solutions of $\mathbf{x}$ lie in the Leaf($\mathbf{x}$), as does Node($\mathbf{x}$).CurrConfig.

The tree $T_{\mathcal{S}}$ is initialized with a user-supplied initial node $\mathbf{x}_{\text{initial}}$ and its inverse kinematics solution $\mathbf{q}_{\text{initial}}$ (Lines 2 and 3 of Algorithm 1). For each iteration of the algorithm, the function ExploreTree() is called, which randomly samples a configuration in the object configuration space $\mathcal{W}$ and finds its nearest node $\mathbf{x}_{\text{n}}$ in $T_{\mathcal{S}}$. The current joint space configuration $\mathbf{q}_{\text{curr}} \in \text{Leaf}(\mathbf{x}_{\text{n}})$ is also identified. A path is then extended from $\mathbf{x}_{\text{n}}$ toward the randomly sampled configuration in $\mathcal{W}$ by an *a priori* specified stepsize, and projected to $\mathcal{S}$; the projected node is denoted $\mathbf{x}_{\text{p}}$.

The planner then determines an inverse kinematics solution for $\mathbf{x}_{\text{p}}$, with $\mathbf{q}_{\text{curr}}$ as its initial guess; the resulting inverse kinematics solution is denoted $\mathbf{q}_{\text{next}}$. A path on $\mathcal{F}$ connecting $\mathbf{q}_{\text{curr}}$ and $\mathbf{q}_{\text{next}}$ is then determined. If the path is collision-free, then the joint space path Q is extended from $\mathbf{q}_{\text{curr}}$ to $\mathbf{q}_{\text{next}}$ (Lines 9 to 12 of Algorithm 1). If however a collision occurs, the object is released and a random search is made for a feasible regrasp configuration $\mathbf{q}_{\text{arrival}}$ on the same leaf Leaf($\mathbf{x}_{\text{n}}$). Once a suitable $\mathbf{q}_{\text{arrival}}$ is found, a jump motion from the current release state $\mathbf{q}_{\text{curr}}$ to the regrasp state $\mathbf{q}_{\text{arrival}}$ is then performed (Lines 14 to 16 of Algorithm 1).

A key step in our planner, and one that turns out to be quite effective in reducing the number of overall release-regrasp operations as verified in our later experiments, is that in searching for $\mathbf{q}_{\text{arrival}}$, we impose the condition that it lie on a disconnected region of the leaf from $\mathbf{q}_{\text{curr}}$. To balance this condition with computational efficiency requirements, we accept the randomly sampled $\mathbf{q}_{\text{arrival}}$ as a valid node only if the path between $\mathbf{q}_{\text{curr}}$ and $\mathbf{q}_{\text{arrival}}$ on $\mathcal{F}$ results in a collision. The planning algorithm then replaces $\mathbf{q}_{\text{curr}}$ by $\mathbf{q}_{\text{arrival}}$ (Line 16). The current configuration and the subsequent path (which can be either a connected path or jump path) is saved in the container $T_{\mathcal{S}}$.Node($\mathbf{x}_{\text{p}}$).CurrConfig and $T_{\mathcal{S}}$.Node($\mathbf{x}_{\text{p}}$).Path, respectively. The above steps are repeated until the Tree $T_{\mathcal{S}}$ reaches $\mathbf{x}_{\text{final}}$, and the final path Q is extracted. The

algorithmic and computational details of our planner are provided in the subsequent sections.

Algorithm 1 Planning Algorithm

1: **Given:** $\mathbf{x}_{\text{initial}}, \mathbf{x}_{\text{final}}$
2: Tree $T_{\mathcal{S}}$.Init($\mathbf{x}_{\text{initial}}$)
3: $T_{\mathcal{S}}$.Node($\mathbf{x}_{\text{initial}}$).CurrConfig ← InverseKinematics($\mathbf{q}_{\text{rand}}, \mathbf{x}_{\text{initial}}$)
4: **repeat**
5: $\mathbf{q}_{\text{curr}}, \mathbf{x}_{\text{n}}, \mathbf{x}_{\text{p}}$ ← ExploreTree($T_{\mathcal{S}}$)
6: $\mathbf{q}_{\text{next}}$ ← InverseKinematics ($\mathbf{q}_{\text{curr}}, \mathbf{x}_{\text{p}}$)
7: $\overline{\mathbf{q}_{\text{curr}}\mathbf{q}_{\text{next}}}$ ← FindPathOn$\mathcal{F}$($\mathbf{q}_{\text{curr}}, \mathbf{q}_{\text{next}}$)
8: **if** CollisionFree($\overline{\mathbf{q}_{\text{curr}}\mathbf{q}_{\text{next}}}$) **then**
9: $T_{\mathcal{S}}$.AddNode($\mathbf{x}_{\text{p}}$)
10: $T_{\mathcal{S}}$.AddEdge($\mathbf{x}_{\text{n}}, \mathbf{x}_{\text{p}}$)
11: $T_{\mathcal{S}}$.Node($\mathbf{x}_{\text{p}}$).Path ← $\overline{\mathbf{q}_{\text{curr}}\mathbf{q}_{\text{next}}}$
12: $T_{\mathcal{S}}$.Node($\mathbf{x}_{\text{p}}$).CurrConfig ← $\mathbf{q}_{\text{next}}$
13: **else**
14: $\overline{\mathbf{q}_{\text{curr}}\mathbf{q}_{\text{arrival}}}$ ← Jump($\mathbf{q}_{\text{curr}}, \mathbf{x}_{\text{n}}, \mathbf{x}_{\text{p}}$)
15: $T_{\mathcal{S}}$.Node($\mathbf{x}_{\text{n}}$).Path ← $\overline{\mathbf{q}_{\text{curr}}\mathbf{q}_{\text{arrival}}}$
16: $T_{\mathcal{S}}$.Node($\mathbf{x}_{\text{n}}$).CurrConfig ← $\mathbf{q}_{\text{arrival}}$
17: **end if**
18: **until** $\mathbf{x}_{\text{final}}$ is added to $T_{\mathcal{S}}$
19: Q ← ExtractPath()
20: **return** final path Q

3.1. ExploreTree()

The function ExploreTree($T_{\mathcal{S}}$) (see Algorithm 2) takes as input the tree $T_{\mathcal{S}}$, and extends the tree as follows: (i) a node $\mathbf{x}_{\text{r}}$ is first randomly sampled in $\mathcal{W}$, and its nearest neighbor on the tree, denoted $\mathbf{x}_{\text{n}}$, is determined. (ii) From $\mathbf{x}_{\text{n}}$, the tree $T_{\mathcal{S}}$ is then extended toward $\mathbf{x}_{\text{r}}$ by a fixed stepsize and then projected back to $\mathcal{S}$. ExploreTree() returns the nodes $\mathbf{x}_{\text{n}}$, $\mathbf{x}_{\text{p}}$, and the current configuration $\mathbf{q}_{\text{curr}}$ in Leaf($\mathbf{x}_{\text{n}}$). To enhance planner efficiency, when $\mathbf{x}_{\text{r}}$ is randomly sampled in line 1 of Algorithm 2, $\mathbf{x}_{\text{final}}$ is substituted in lieu of $\mathbf{x}_{\text{r}}$ by probability p_{final}. The function Distance($\mathbf{x}_1, \mathbf{x}_2$) in line 4 returns the Euclidean distance between $\mathbf{x}_1$ and $\mathbf{x}_2$.

Algorithm 2 ExploreTree($T_{\mathcal{S}}$)

1: $\mathbf{x}_{\text{r}}$ ← RandomSample($\mathcal{W}$)
2: $\mathbf{x}_{\text{n}}$ ← NearestNeighbor($T_{\mathcal{S}}, \mathbf{x}_{\text{r}}$)
3: $\mathbf{x}_{\text{p}}$ ← ConstrainedExtend($\mathbf{x}_{\text{n}}, \mathbf{x}_{\text{r}}$, stepsize)
4: **if** Distance($\mathbf{x}_{\text{n}}, \mathbf{x}_{\text{final}}$) < stepsize **then**
5: $\mathbf{x}_{\text{p}} \leftarrow \mathbf{x}_{\text{final}}$
6: **end if**
7: $\mathbf{q}_{\text{curr}}$ ← $T_{\mathcal{S}}$.Node($\mathbf{x}_{\text{n}}$).CurrConfig
8: **return** $\mathbf{q}_{\text{curr}}, \mathbf{x}_{\text{n}}, \mathbf{x}_{\text{p}}$

Algorithm 3 ConstrainedExtend($\mathbf{x}_{\text{n}}, \mathbf{x}_{\text{r}}$, stepsize)

1: $\mathbf{x}_{\text{p}} \leftarrow \mathbf{x}_{\text{n}} + \min\,(\text{stepsize}, \| \mathbf{x}_{\text{p}} - \mathbf{x}_{\text{n}} \|) \frac{\mathbf{x}_{\text{p}} - \mathbf{x}_{\text{n}}}{\|\mathbf{x}_{\text{p}} - \mathbf{x}_{\text{n}}\|}$
2: $\mathbf{x}_{\text{p}}$ ← ProjectTo$\mathcal{S}$($\mathbf{x}_{\text{p}}$)
3: **return** $\mathbf{x}_{\text{p}}$

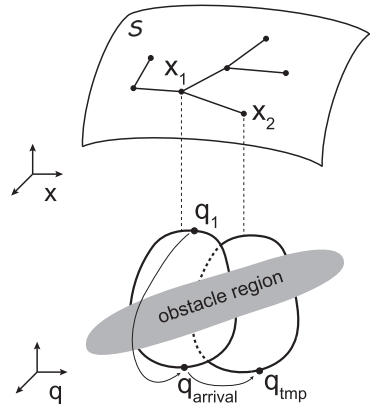

Figure 4. Illustration of Jump().

3.2. Jump()

The function Jump($\mathbf{q}_1, \mathbf{x}_1, \mathbf{x}_2$) takes as inputs the nodes $\mathbf{x}_1$ and $\mathbf{x}_2$ of $T_{\mathcal{S}}$, and the joint configuration $\mathbf{q}_1$, where $\mathbf{q}_1$ is an inverse kinematics solution of $\mathbf{x}_1$. The function then randomly generates a node $\mathbf{q}_{\text{arrival}}$ that is another inverse kinematics solution to $\mathbf{x}_1$, and returns a collision-free path $\overline{\mathbf{q}_1\mathbf{q}_{\text{arrival}}}$. Figure 4 illustrates the Jump() function within the leaf corresponding to $\mathbf{x}_1$. The three conditions described in line 5 of Algorithm 4 must all be satisfied in order to exit the loop; the user-supplied collision checking function CollisionFree() takes as input a configuration or a path segment, and returns *true* if no collisions occur. The function FindPathIn$\mathcal{M}$() in line 6 returns a path in the ambient joint configuration space that connects $\mathbf{q}_1$ and $\mathbf{q}_{\text{arrival}}$ using any basic bi-directional RRT method as described in, e.g. [10].

Algorithm 4 Jump($\mathbf{q}_1, \mathbf{x}_1, \mathbf{x}_2$)

1: **repeat**
2: $\mathbf{q}_{\text{arrival}}$ ← InverseKinematics($\mathbf{q}_{\text{rand}}, \mathbf{x}_1$)
3: $\mathbf{q}_{\text{tmp}}$ ← InverseKinematics($\mathbf{q}_{\text{arrival}}, \mathbf{x}_2$)
4: $\overline{\mathbf{q}_1\mathbf{q}_{\text{arrival}}}$ ← FindPathOn$\mathcal{F}$($\mathbf{q}_1, \mathbf{q}_{\text{arrival}}$)
5: **until** CollisionFree($\mathbf{q}_{\text{arrival}}$) AND
CollisionFree($\mathbf{q}_{\text{tmp}}$) AND
NOT CollisionFree($\overline{\mathbf{q}_1\mathbf{q}_{\text{arrival}}}$)
6: $\overline{\mathbf{q}_1\mathbf{q}_{\text{arrival}}}$ ← FindPathIn$\mathcal{M}$($\mathbf{q}_1, \mathbf{q}_{\text{arrival}}$)
7: **return** $\overline{\mathbf{q}_1\mathbf{q}_{\text{arrival}}}$

3.3. FindPathOn$\mathcal{F}$()

The function FindPathOn$\mathcal{F}$($\mathbf{q}_1, \mathbf{q}_2$) takes as input two configurations $\mathbf{q}_1$ and $\mathbf{q}_2$ on the foliation $\mathcal{F}$, and returns a path from $\mathbf{q}_1$ to $\mathbf{q}_2$ that lies on the $\mathcal{F}$. First, it uniformly discretizes the line from $\mathbf{q}_1$ to $\mathbf{q}_2$ by a pre-defined unit interval length d_{step}. It projects all configurations to the foliation $\mathcal{F}$, then returns the projected configurations as a path (see Algorithm 5).

Algorithm 5 FindPathOn$\mathcal{F}(\mathbf{q}_1, \mathbf{q}_2)$

1: path($\mathbf{q}_1, \mathbf{q}_2$)[ ]
2: path($\mathbf{q}_1, \mathbf{q}_2$).Add($\mathbf{q}_1$)
3: $i \leftarrow 1$
4: **repeat**
5: $\quad \mathbf{q}_{\text{tmp}} \leftarrow \mathbf{q}_1 + i \times (\mathbf{q}_2 - \mathbf{q}_1)/ \parallel \mathbf{q}_2 - \mathbf{q}_1 \parallel \times d_{\text{step}}$
6: $\quad \mathbf{q}_{\text{tmp}} \leftarrow$ ProjectTo$\mathcal{F}(\mathbf{q}_{\text{tmp}})$
7: $\quad$ path($\mathbf{q}_1, \mathbf{q}_2$).Add($\mathbf{q}_{\text{tmp}}$)
8: $\quad i++$
9: **until** $i > \parallel \mathbf{q}_2 - \mathbf{q}_1 \parallel / d_{\text{step}}$
10: path($\mathbf{q}_1, \mathbf{q}_2$).Add($\mathbf{q}_2$)
11: **return** path($\mathbf{q}_1, \mathbf{q}_2$)

3.4. Projection operations

The algorithm uses two types of projections:

(i) projection to the task constraint manifold $\mathcal{S}$;
(ii) projection to the foliation $\mathcal{F}$.

For each of the two cases, we first define the error vector $e = g(\theta)$ and then perform the relevant projection operation by finding the parameter value θ that minimizes the error e. We do so via a simple pseudo-inverse-based least squares method: expanding $g(\theta)$ to first-order as

$$e + \delta e \simeq g(\theta) + \frac{\partial g}{\partial \theta}(\theta)\delta\theta, \tag{1}$$

we apply the iteration

$$\theta_{\text{new}} \leftarrow \theta - J(\theta)^{\dagger}\delta e \tag{2}$$

where $J(\theta) = \frac{\partial g}{\partial \theta}(\theta)$ and $J(\theta)^{\dagger}$ denotes the pseudo-inverse of $J(\theta)$:

$$J(\theta)^{\dagger} = J(\theta)^{\mathrm{T}}[J(\theta)J(\theta)^{\mathrm{T}}]^{-1} \tag{3}$$

3.4.1. Projection to the task constraint manifold $\mathcal{S}$

ProjectTo$\mathcal{S}(\mathbf{x})$ is used in Algorithm 3 when the tree $T_{\mathcal{S}}$ extends on the constraint manifold. We define the error vector $e = g(\theta)$ with parameter $\theta = \mathbf{x}$ as

$$g(\mathbf{x}) = \text{Distance}(\mathbf{x}, \mathcal{S}). \tag{4}$$

For calculating Distance($\mathbf{x}, \mathcal{S}$), we find the nearest point in $\mathcal{S}$ from $\mathbf{x}$ and return the Euclidean distance from the nearest point to $\mathbf{x}$.

3.4.2. Projection to the foliation $\mathcal{F}$

Projection to the foliation $\mathcal{F}$, expressed as ProjectTo$\mathcal{F}(\mathbf{q})$, is used in the function FindPathOn$\mathcal{F}()$ in Algorithm 5 and also in the sampling procedures of topological exploration stage described later in Section 4.1. The error vector $e = g(\theta)$ with parameter $\theta = \mathbf{q}$ is defined as

$$g(\mathbf{q}) = \text{Distance}(f(\mathbf{q}), \mathcal{S}) \tag{5}$$

where $f(\mathbf{q})$ is the forward kinematics of robot. The distance is calculated identically to the task constraint manifold projection procedure described above.

3.5. Inverse kinematics

The inverse kinematics function InverseKinematics ($\mathbf{q}_{\text{init}}, \mathbf{x}$) takes as input a desired position $\mathbf{x}$ of the end-effector, together with an initial guess $\mathbf{q}_{\text{init}}$ for its solution. It then calculates an inverse kinematics solution $\mathbf{q} \in \mathcal{M}$ that satisfies $f(\mathbf{q}) = \mathbf{x}$. If the robot is kinematically redundant, an infinite number of solutions may exist. In our implementation we use a standard pseudo-inverse-based least squares method, although in principle one can use any number of numerical inverse kinematics iterative algorithms, e.g. Chapter 11 of [11].

4. Path refinement

The solution trajectory calculated in Section 3 contains jump motions, i.e. when a robot releases and regrasps an object. Because of the stochastic nature of the algorithm, not all of the jump motions may be necessary. The purpose of path refinement is to reduce unnecessary jump motions, and is composed of two stages: (i) topological exploration, and (ii) eliminating unnecessary jump motions. In the topological exploration stage, the number of disconnected pieces of the foliation $\mathcal{F}$ is estimated via a k-nearest neighbor clustering method graph search. Unnecessary jump motions are then eliminated based on information obtained about the topological structure of $\mathcal{F}$.

4.1. Topological exploration

In the topological exploration stage, the number of disconnected pieces in the foliation $\mathcal{F}$ is estimated via sampling. Denoting by m the dimension of $\mathcal{M}$, we first randomly sample n points in $\mathcal{M}$, and then project these to $\mathcal{F}$; the coordinates of these n projected points are then arranged as rows of the $n \times m$ matrix $X = \{x_{ij}\}$. In Figure 5(a), the sampled data X are shown for the illustrative example. We then use k-nearest neighbor clustering to construct an undirected graph with n nodes. Details of how to find the optimal number $\hat{K}$ of disconnected components of $\mathcal{F}$, and also how to partition X into $\hat{K}$ clusters, are described in Algorithm 6.

The undirected graph UG is constructed in line 1 with n nodes and no edges; UG.edge is initialized as an $n \times n$ matrix with all components zero. The function ForwardKinematics() in line 2 of Algorithm 6, which takes the data matrix X as input, calculates the task space Cartesian positions of all of the robot's joints, and arranges them

Algorithm 6 Topological Exploration

```
1: UG.initialize(n)
2: Y ← ForwardKinematics(X)
3: for i = 1 to n do
4:     idxs ← k-NN(y_i, Y)
5:     for j = 1 to k do
6:         UG.edge(i, idxs(j)) ← 1
7:     end for
8: end for
9: K̂ ← depthFirstSearch(UG)
10: return K̂
```

in the form of an $n \times 3m$ matrix Y. k-NN(y_i, Y) in line 4 finds k nearest neighbors of y_i in the set Y with respect to the Euclidean distance metric, and returns their indices, i.e. y_i is the ith row of Y. The edge between the ith node and the idxs(j)-th node is connected in lines 5–7. The constructed undirected graph is searched by a depth-first search method to find the connected components of UG; the number of connected components $\hat{K}$ is then obtained. As can be seen in Figure 5, the sampled data X in (a) are clustered into two clusters as shown in (b), marked by 'x' and 'o'.

4.2. Reducing the number of jumps

Exploiting knowledge about the estimated number $\hat{K}$ of disconnected pieces of the foliation $\mathcal{F}$ and the clustered data X (or Y), we now attempt to reduce the number of unnecessary jumps from the solution path found in Section 3. The algorithm proceeds as follows:

(1) Every configuration in X (or Y) is first assigned to its own index e.g. as shown in Figure 5.
(2) For all departure and arrival configurations of jump paths from the solution path, calculate the forward kinematics and obtain the task space Cartesian coordinates for each of the joints, find the nearest set of values in the data-set Y, and assign the same cluster index as that of its nearest configuration.
(3) If the assigned cluster indices of the departure and arrival configurations of the jump paths are repeated, two departure and arrival configurations that have the same index can be connected on the foliation $\mathcal{F}$.
(4) Try to find any connected path on $\mathcal{F}$ between the two configurations.
(5) Eliminate jump paths and replace them with connected paths.

In the fourth step, connected paths on the foliation $\mathcal{F}$ can be found in a number of ways. For our purposes we adopt the following sample-and-project method: (i) a random configuration in $\mathcal{M}$ is sampled; (ii) the tree node that is nearest to the randomly sampled configuration is identified; (iii) the tree is extended from the nearest node to the randomly sampled configuration by an *a priori*-specified stepsize; (iv) the extended node is projected back to the foliation $\mathcal{F}$. An example in Section 5.4 outlines the details of this procedure.

5. Experiments

In this section, we perform numerical experiments to evaluate the performance of our proposed algorithm. All simulations are performed using MATLAB running on an Intel Core2 Quad processor Q9450, @ 2.66GHz, and 5.0GB RAM on Windows 8. Using computation times, the number of projections, the number of jump motions, and the path length in the configuration space as performance criteria, we compare our algorithm with the $\mathbb{T}$-based MMP-UE method described in [8] and also to Simeon *et al*'s approach.[4] The path length l is calculated from the connected paths $\{Q_c^1, \cdots, Q_c^{N+1}\}$ as described by Equation (6):

$$l = \sum_{i}^{N+1} \sum_{j}^{n_i-1} \sum_{k}^{m} \mid Q_c^i(j+1,k) - Q_c^i(j,k) \mid \qquad (6)$$

where N is the number of jump paths, n_i is the number of configurations in the connected path Q_c^i, m is the dimension of the joint configuration space, and $Q_c^i(j,k)$ is the jth configuration and kth joint value of Q_c^i. All the reported mean and standard deviation values reported in Tables 1–5 are obtained by averaging over 10 trials. For all simulations, the parameter p_{final} is set to 0.15, d_{step} (Algorithm 5) is set to 0.01, and the stepsizes in $T_{\mathcal{S}}$ (Algorithm 2), FindPathIn$\mathcal{M}$() (Algorithm 4), and connected path finding algorithm (Section 4.2) are all set to 0.1.

5.1. Three-DOF planar open chain

5.1.1. One circular obstacle

In this example, we consider a three-DOF planar open chain whose links are all of unit length and whose base is placed at the origin of the x–y plane. A circular obstacle of radius 0.8 is placed in the workspace with center at coordinates (1.0, 0,0) as shown in Figure 6. The robot end-effector must transport a point mass object along a vertical linear path beginning at coordinates (2.0, 2.0) and ending at (2.0, -2.0).

The $\mathbb{T}$-planner parameter n_c shown in Table 1 represents the number of sampled configurations in a single leaf. For our experiments we vary its value from 10 to 50. From the results shown in Table 1, it can be seen that our proposed planner outperforms other planners, especially with respect to execution time. The clear reason is the smaller number of projections compared to

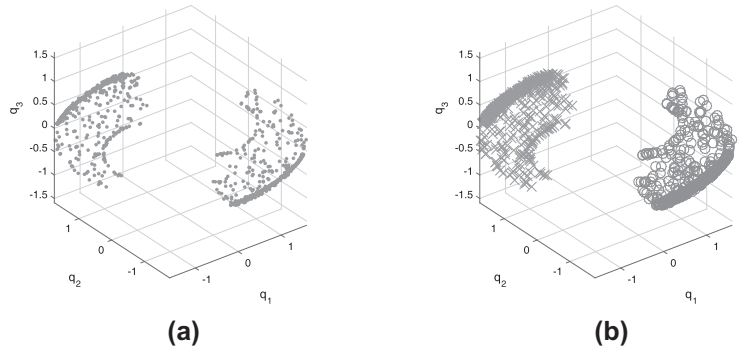

Figure 5. Topological exploration. (a) Sampled data, $X = \{x_{ij}\}$, $n = 10,000$. (b) Clustered data, $\hat{K} = 2$.

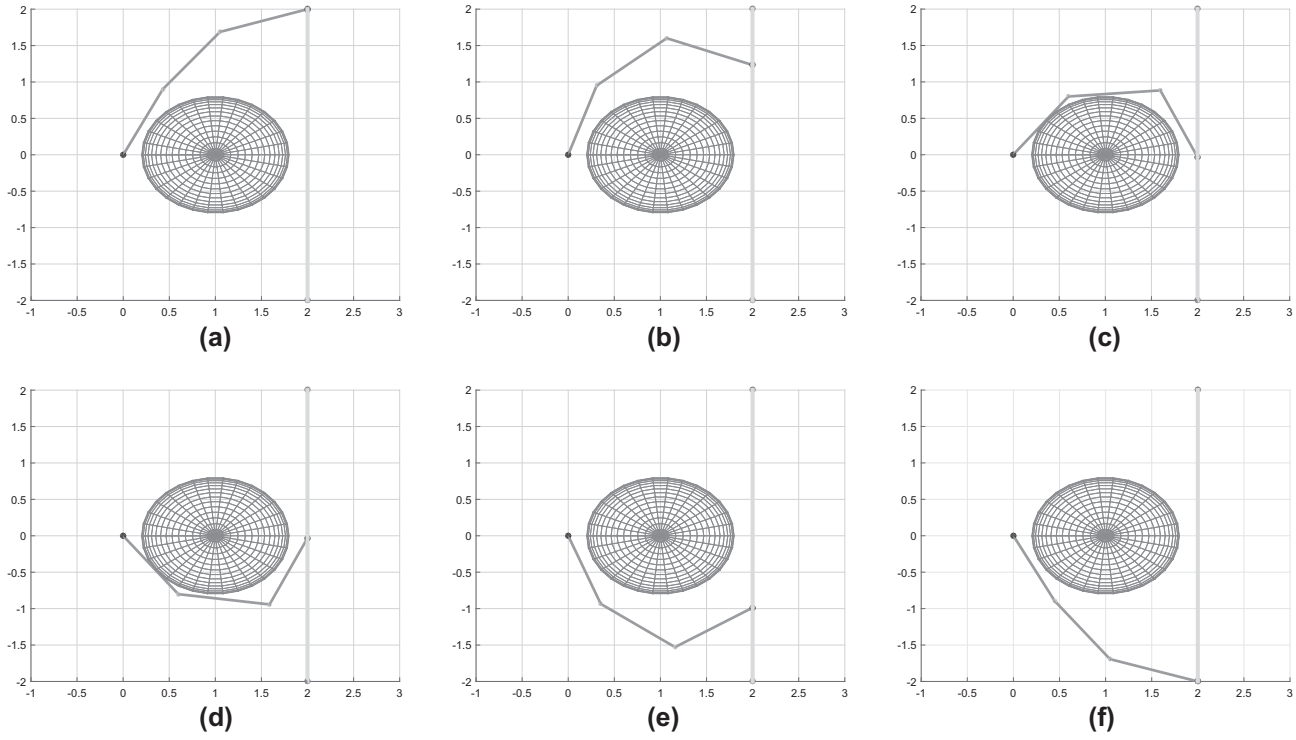

Figure 6. Simulation results of 3 DOF planar manipulator with 1 obstacle.

other planners. For the $\mathbb{T}$-planner, as n_c is increased the path length decreases while the execution time increases. Figure 6 graphically displays the motion of the manipulator: (a) and (f) are the initial and final positions, respectively, while (c) and (d) are the release and re-grasp configurations for the object, respectively. Figure 5 shows the sampled and clustered data for this case.

5.1.2. Two circular obstacles

In this example, we consider the same three-link planar open chain and linear path, but with two circular obstacles: both are of radius 0.3 and centered at coordinates (1.25, 1.0) and (1.25, −1.0), respectively. The mean and standard deviation values for this experiment (averaged over 10 trials as before) are shown in Table 2. Compared to other methods, our planner finds a solution path in considerably less time – other methods take anywhere from 10 to 50 times longer than our method, which takes an average of 2.78 s to find a path. The final

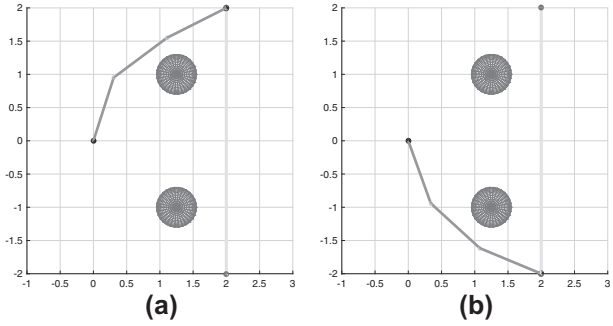

Figure 7. Initial and final postures for a 3-DOF planar open chain with two obstacles.

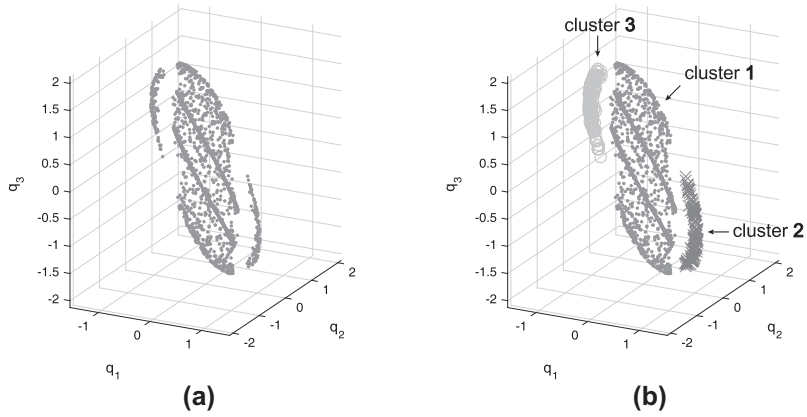

Figure 8. Topological exploration for the example in Section 5.1.2. (a) Sampled data, $X = \{x_{ij}\}$, $n = 10,000$. (b) Clustered data, $\hat{K} = 3$.

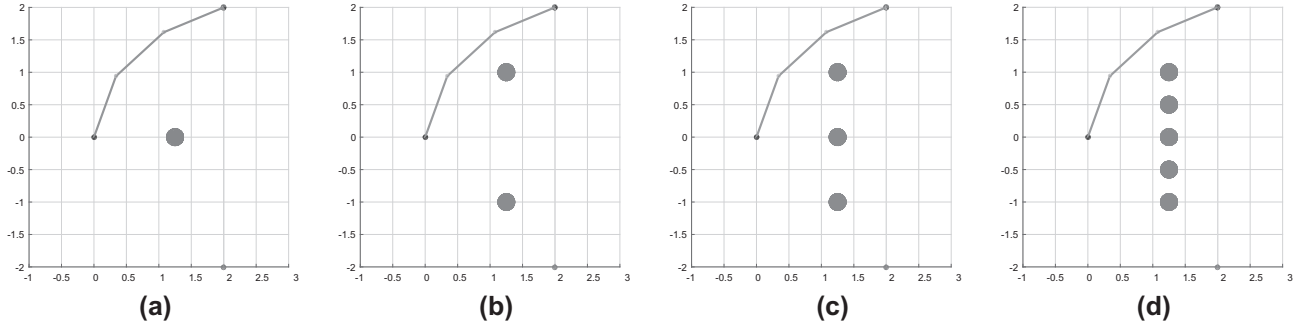

Figure 9. Initial postures for variable number of obstacles. (a) One obstacle. (b) Two obstacles. (c) Three obstacles. (d) Five obstacles.

Table 1. Results for 3-DOF planar open chain with one obstacle.

		Time (sec)	# Projections	# Jumps	Path length
Our algorithm		1.10 ± 0.36	701.00 ± 146.08	1.00 ± 0.00	6.58 ± 1.93
	n_c=10	42.67 ± 4.96	8010.30 ± 990.00	1.00 ± 0.00	13.92 ± 2.45
$\mathbb{T}$-planner [8]	n_c=20	79.27 ± 6.54	13920.80 ± 1027.18	1.00 ± 0.00	9.31 ± 1.52
	n_c=50	191.43 ± 18.65	32157.30 ± 2933.54	1.00 ± 0.00	6.82 ± 0.88
Simeon et al. [4]		80.13 ± 156.47	13252.00 ± 27800.71	1.00 ± 0.00	9.54 ± 2.37

path length is shorter – other methods produce paths that are anywhere from 2 to 5 times longer than our method, which produces an average path length of 5.88. The average number of jumps, however, is slightly higher for our method (2.8) than that of other methods (2–2.7).

Results obtained using the $\mathbb{T}$-planner are similar to those obtained for the previous example, taken over a range of values for the parameter n_c. Using Simeon *et al*'s probabilistic roadmap-based approach,[4] we find that the computation times are considerably longer, primarily because of the large number of projections needed to construct a topological graph. Figure 7(a) and (b), respectively, show the initial and final positions for a typical sample trajectory. The sampled data and clustered results are shown in Figure 8.

5.1.3. *Effect of obstacles*

In this example, we consider the same three-link planar open chain and linear path in task space, but in order to assess the effect of obstacles on algorithm performance, we vary the number of obstacles from one to five. The obstacle positions are shown in Figure 9: a circular obstacle is centered at (1.25, 0.0), and the other circular obstacles are centered at equally spaced intervals between (1.25, 1.0) and (1.25, -1.0); all circular obstacles are of radius 0.13. From Table 3 it can be seen that planning times increase with the number of obstacles; this can be attributed to the increase in the number of projections. The computation times and number of projections are similar for the three and five obstacle cases, most likely because the volumes occupied by the obstacles in the joint

Table 2. Results for 3-DOF planar open chain with two obstacles.

		Time (sec)	# Projections	# Jumps	Path length
Our algorithm		2.78 ± 1.52	1020.90 ± 258.33	2.80 ± 0.60	5.88 ± 1.38
	n_c=10	25.01 ± 3.46	6414.60 ± 809.37	2.50 ± 0.67	32.16 ± 3.77
$\mathbb{T}$-planner [8]	n_c=20	40.72 ± 4.10	7697.20 ± 583.26	2.00 ± 0.00	19.96 ± 2.50
	n_c=50	99.01 ± 12.32	15341.20 ± 1724.18	2.50 ± 0.67	11.96 ± 1.88
Simeon et al. [4]		152.64 ± 114.19	22470.80 ± 17219	2.70 ± 0.64	15.77 ± 4.14

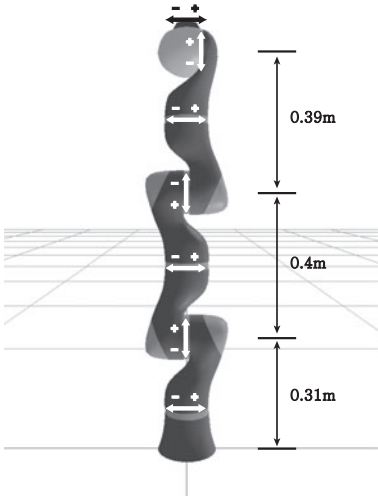

Figure 10. KUKA lightweight robot (LWR).

configuration space are nearly the same. The number of jumps also increases with the number of obstacles, most likely because the number of disconnected pieces of the foliation $\mathcal{F}$ increases. For this particular case study, however, the size of the obstacles does not seem to have as much influence as the number of obstacles.

5.2. KUKA robot

In this example, we consider the seven-DOF KUKA robot shown in Figure 10. The task objective is to move the disk-shaped object from the initial configuration $(-0.4, -0.4, 0.735)$ to the final configuration $(0.4, 0.4, 0.735)$, in such a way that the object must always remain in contact with the planar surface defined by $z = 0.735$. A box-like obstacle is placed in the center of its workspace. The obstacle is placed such that two of its diagonally opposite vertices are located at $(-0.04, -0.5, 0.534)$ and $(0.04, 0.5, 0.574)$. The kinematic structure of the arm and the nature of the task makes it impossible for the robot to guide the object from the initial to final configuration in a single continuous motion; at some arm configuration, the robot must release the object, then regrasp the object at a different arm configuration.

The results of our experiments are shown in Table 4. Compared to the $\mathbb{T}$-planner, our planner finds better quality solutions – measured with respect to the number of jumps and path length – more quickly. The $\mathbb{T}$-planner produces paths whose number of jumps decreases as n_c increases, since a large number of samples in a single leaf leads to a higher probability of finding connected paths. However, computation times increase because of the larger number of projections with growing n_c. Simeon *et al*'s approach produces paths with fewer jumps than our approach, but average execution times and path lengths are considerably longer, primarily because of the inherently larger computational overhead of roadmap-based methods. Figure 11 shows a typical trajectory for the robot: here (a) and (d) are the initial and final positions, respectively, while (b) and (c) are, respectively, the release and regrasp configurations of the object.

5.3. Jungle gym experiments

We now perform a set of experiments involving a redundant open chain pushing an object through a uniform three-dimensional spatial grid, or a traditional jungle gym. In the first experiment, we consider an 8-DOF open chain consisting of four unit-length links serially connected by universal joints, pushing an object along a linear path through the center of a $1 \times 1 \times 1$ grid as shown in Figure 12. In the second experiment, we consider a 10-DOF open chain consisting of five unit-length links serially connected by universal joints, pushing an object along a linear path through the center of the bottom layer of a $3 \times 3 \times 3$ grid as shown in (see Figure 13).

The initial and final arm postures for the two experiments are, respectively, shown in (a) and (b) of Figures 12 and 13. Results of our numerical experiments are given in Table 5. For the $1 \times 1 \times 1$ grid, our algorithm finds a considerably shorter path (of average length 6.06) in less than 90 s on average, involving an average of 3.6 jumps and 1294.4 projections. By way of contrast, the $\mathbb{T}$-planner with n_c set to 10 takes an average of 4082 s to produce a path of length 424.48, involving an average of 21.7 jump motions and 54952.4 projections. The results with n_c set to 20 are similar.

The gap in performance is even greater for the $3 \times 3 \times 3$ grid: our algorithm takes an average of 151.68 s to produce paths of average length 5.22, with 17.1 jumps

Table 3. Results for effect of obstacles experiment.

	# Obstacles	Time (sec)	# Projections	# Jumps	Path length
Our algorithm	1	0.59 ± 0.07	529.60 ± 40.46	1.00 ± 0.00	4.36 ± 0.43
	2	2.84 ± 1.49	950.80 ± 188.49	2.40 ± 0.66	5.03 ± 1.74
	3	4.02 ± 2.06	1193.00 ± 292.05	3.80 ± 0.87	6.48 ± 2.40
	5	4.10 ± 0.45	1270.80 ± 139.76	7.70 ± 1.55	4.27 ± 0.51

Table 4. Experimental results for KUKA robot.

		Time (sec)	# Projections	# Jumps	Path length
Our algorithm		27.92 ± 10.06	1775.50 ± 441.57	3.40 ± 0.92	8.54 ± 3.23
$\mathbb{T}$-planner [8]	n_c=10	1060.89 ± 287.87	22908.20 ± 5946.45	9.20 ± 2.89	96.43 ± 10.33
	n_c=20	1946.79 ± 249.69	23869.10 ± 3207.17	5.90 ± 1.76	80.12 ± 6.31
	n_c=50	4644.02 ± 444.28	38047.80 ± 5037.11	3.40 ± 1.11	56.72 ± 6.71
Simeon et al. [4]		2386.43 ± 2311.98	204467.40 ± 191418.80	2.10 ± 0.70	32.03 ± 7.07

Table 5. Results for jungle gym experiments.

			Time (sec)	# Projections	# Jumps	Path length
1 × 1 × 1 grid	Our algorithm		89.64 ± 48.6	1294.40 ± 381.90	3.60 ± 1.50	6.06 ± 3.78
	$\mathbb{T}$-planner [8]	n_c=10	4082.41 ± 844.84	54952.40 ± 8030.55	21.70 ± 2.79	424.48 ± 58.55
		n_c=20	3046.81 ± 720.58	46324.90 ± 10855.85	22.20 ± 2.14	382.73 ± 54.41
3 × 3 × 3 grid	Our algorithm		151.68 ± 186.38	1351.80 ± 474.76	17.10 ± 3.78	5.22 ± 0.52

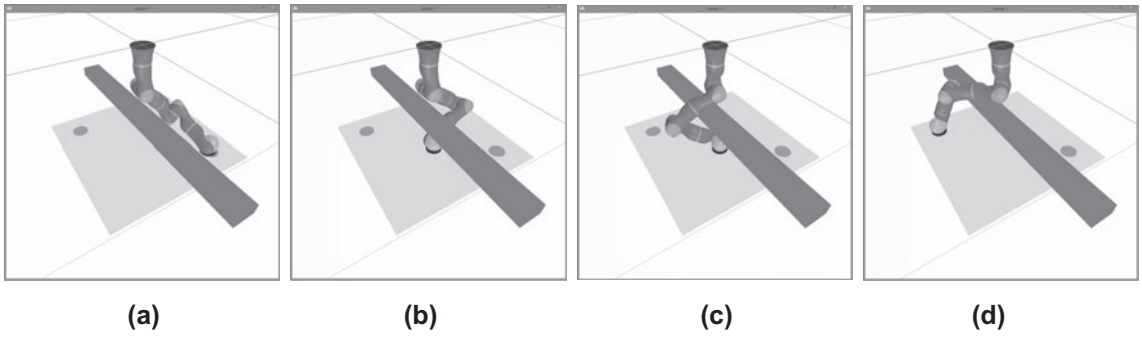

Figure 11. Simulation results for KUKA robot.

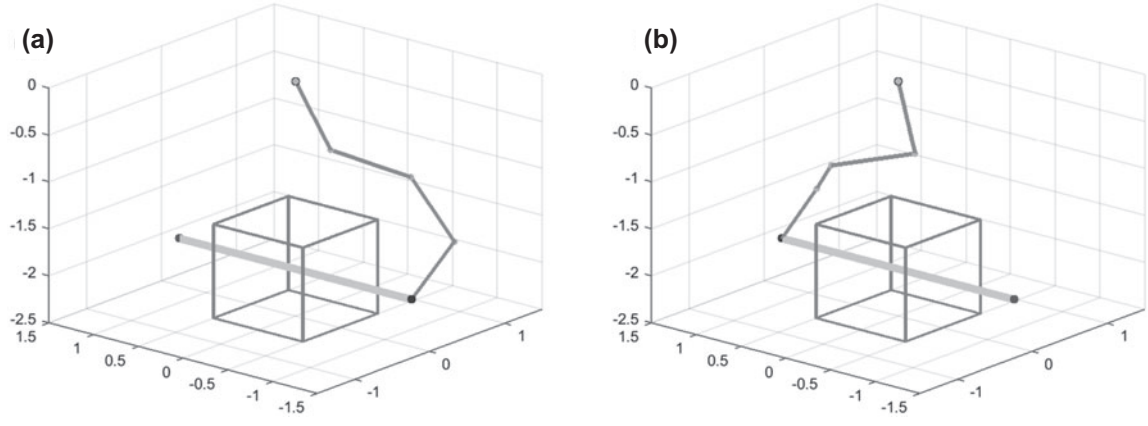

Figure 12. Simulation results for 1 × 1 × 1 grid.

on average and involving 1351.8 projections. In contrast, both the $\mathbb{T}$-planner and the method of Simeon *et al* are unable to find a solution trajectory even after running for several hours.

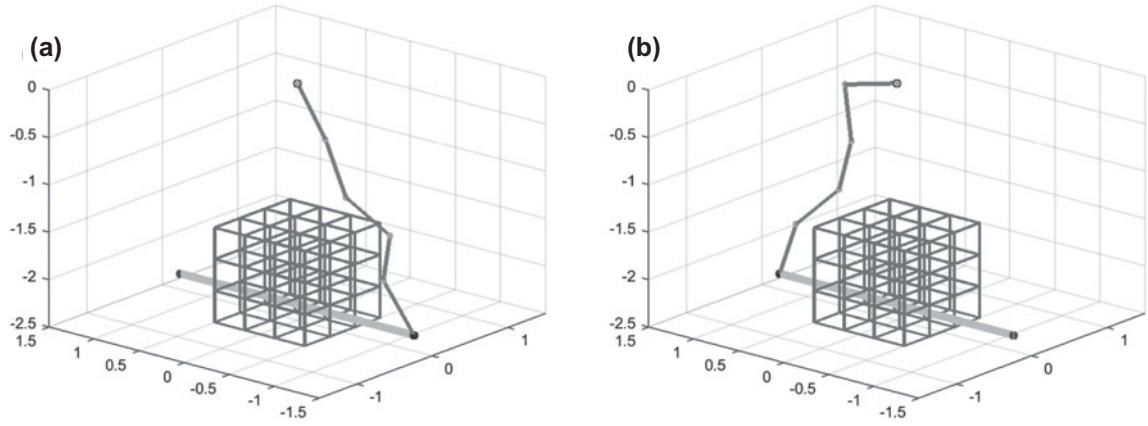

Figure 13. Simulation results for 3 × 3 × 3 grid.

Table 6. Path refinement example.

	Departure	Arrival
Jump motion 1	2	1
Jump motion 2	1	1
Jump motion 3	1	3

Table 7. Results for path refinement.

	3-DOF 1 obstacle	3-DOF 2 obstacles	KUKA robot
Sampling time (sec)	419.50	158.03	1144.94
Exploration time (sec)	21.83	22.31	47.18
Connection time (sec)	–	30.15	1653.55
Total time (sec)	436.29	232.04	533.03
Estimated $\hat{K}$	2	3	2
Original number of jumps	1.00	2.80	3.40
Reduced number of jumps	1.00	2.00	2.20

5.4. Path refinement experiments

We now perform a set of experiments to assess the effectiveness of our path refinement algorithm in reducing the overall number of jump motions (or equivalently, the number of release-regrasp sequences). To intuitively explain the algorithm, we consider the previous example of the 3-DOF planar open chain moving in the presence of two circular obstacles and assume that estimated number $\hat{K} = 3$ of disconnected pieces of the foliation $\mathcal{F}$ and the clustered data X are given as seen in Figure 8(b). In the numerical experiments, three jump motions are produced, and assigned cluster indices for six configurations – three departure configurations and three arrival configurations – are obtained as listed in Table 6. Jump motion 2 is unnecessary because it occurs within the connected piece of the foliation (the middle piece in Figure 8(b)). For this example, our path refinement procedure removes jump motion 2 by connecting the departure and the arrival configurations for jump motion 2. In our subsequent examples involving topological exploration of $\mathcal{F}$, the number of samples n is set to 10,000 while in the k-nearest neighbor clustering algorithm we set k to 10.

Results of the path refinement procedure applied to our solution paths are shown in Table 7. The sampling time is the time required to sample 10,000 configurations on the disconnected foliation, while the exploration time is the time required to estimate the number of pieces of the disconnected foliation. The connection time is the time needed to assign cluster indices for departure and arrival configurations, to determine which configurations should be connected, and to find connected paths to replace jump paths for 10 solution paths for each experiment. $\hat{K}$ is the estimated number of pieces of the disconnected foliation. The (original/reduced) number of jumps is the averaged value of the number of jump motions over 10 trials in the solution path (before/after) applying the path refinement algorithm. For the 3-DOF planar chain with one obstacle, our solution paths do not contain any unnecessary jump motions that can be eliminated. For the 3-DOF planar chain with two obstacles example, the path refinement procedure successfully

removes all unnecessary jump motions, and retains only the minimal required number of jumps. For the KUKA robot example, the estimated number $\hat{K}$ for the disconnected manifold is two, since we use the distance metric in the k-nearest neighbor algorithm by considering the task space Cartesian positions of the joints. The constraint manifold, however, can be disconnected into more than two pieces as a result of obstacles, various kinematic constraints, and joint limits. In this example, therefore, our path refinement algorithm reduces the number of jumps from 3.4 to 2.2, but is unable to attain its minimal value of one. For the jungle gym experiments, our path refinement procedure is unable to determine the optimal $\hat{K}$, most likely because of the high dimensionality of the data.

6. Conclusion

This paper has presented a new randomized path planning algorithm for tasks requiring the release and regrasp of objects. Such problems are formulated as a planning problem on a foliated manifold, with the foliated manifold partitioned into different topological components if obstacles are present in the workspace. A distinctive feature of our algorithm vis-á-vis existing algorithms is the significant improvement in computational efficiency, achieved primarily as a result of the reduced number of projection to the foliated manifold $\mathcal{F}$. Although probabilistic completeness of our algorithm cannot be guaranteed at present, because our planning algorithm samples in both the task and configuration spaces, in all of our experimental case studies, our algorithm successfully finds a feasible solution path when one exists.

A second contribution of our planner is a post-processing path refinement stage, in which the topological structure of $\mathcal{F}$ is learned via a sampling-based k-nearest neighbor clustering method, followed by a graph search to reduce the number of release-regrasp motions. Extensive numerical experiments with our planner demonstrate that our planner finds solution paths more quickly than existing methods, and that the resulting paths are typically of shorter length and contain fewer jumps. The performance advantages of our planner appear to be even greater for more complex environments. Our path refinement stage can be applied to any generic path planning algorithm for tasks that require the release and regrasp of objects. One way to improve performance is to develop a distance metric for the k-nearest neighbor method that considers not only the task space, but also the joint configuration space, to more accurately estimate the number of pieces of the disconnected manifold.

Note

1. Depending on the endpoint location, there may be up to two closed path solutions.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported in part by the BMRR center funded by Defense Acquisition Program Administration [UD130070ID], Technology Innovation Program funded by the Ministry of Trade, Industry & Energy [10048320], SNU-IAMD, and SNU-MAE BK21+.

Supplementary material

The supplementary material for this paper is available online at http://dx.doi.org/10.1080/01691864.2015.1105150.

Notes on contributors

Jinkyu Kim received BS and MS degrees in mechanical and aerospace engineering from Seoul National University in 2011 and 2013, respectively. He is currently pursuing PhD degree at SNU. His research interests are in robot motion planning and control, and motion optimization.

Inyoung Ko received the BS degree in 2008 and the PhD degree in 2014, both in Mechanical and Aerospace Engineering from Seoul National University. His interests are in robot motion planning and control, mobile manipulation, and industrial robots. Since 2014, he is working at Samsung Electronics.

Frank C. Park received his BS in electrical engineering from MIT in 1985, and PhD in applied mathematics from Harvard University in 1991. From 1991 to 1995 he was assistant professor of mechanical and aerospace engineering at the University of California, Irvine. Since 1995 he has been professor of mechanical and aerospace engineering at Seoul National University. His research interests are in robot mechanics, planning and control, vision and image processing, and related areas of applied mathematics. He has been an IEEE Robotics and Automation Society Distinguished Lecturer, and received best paper awards for his work on visual tracking and parallel robot design. He

has served on the editorial boards of the Springer Handbook of Robotics, Springer Advanced Tracts in Robotics (STAR), Robotica, and the ASME Journal of Mechanisms and Robotics. He has held adjunct faculty positions at the NYU Courant Institute and the Interactive Computing Department at Georgia Tech. He is a fellow of the IEEE, current editor-in-chief of the IEEE Transactions on Robotics, and developer of the EDX course Robot Mechanics and Control I, II.

References

[1] Berenson D, Srinivasa S, Kuffner J. task space regions: a framework for pose-constrained manipulation planning. Int. J. Robot. Res. 2011;30:1435–1460.

[2] Kim B, Um TT, Suh C, et al. Tangent bundle RRT: a randomized algorithm for constrained motion planning. Robotica. 2016;34:202–225.

[3] Oriolo G, Vendittelli M. A control-based approach to task-constrained motion planning. In: Proceedings of IEEE/RSJ intelligent robots and systems; St. Louis, MO, USA; 2009. p. 297–302.

[4] Siméon T, Laumond JP, Cortés J, Sahbani A. Manipulation planning with probabilistic roadmaps. Int. J. Robot Res. 2004;23:729–746.

[5] Kavraki L, Latombe JC. Randomized preprocessing of configuration for fast path planning. In: Proceedings of IEEE international conference on robotics and automation; San Diego, CA, USA; 1994. p.2138–2145.

[6] Harada K, Tsuji T, Laumond JP. A manipulation motion planner for dual-arm industrial manipulators. In: Proceedings of IEEE international conference on robotics and automation; Hong Kong, China; 2014. p. 928–934.

[7] Saut JP, Gharbi M, Cortés J, et al. Planning pick-and-place tasks with two-hand regrasping. In: Proceedings of IEEE/RSJ intelligent robots and systems; Taipei, Taiwan; 2010. p. 4528–4533.

[8] Kim JT, Kim D. Manipulator motion planning with unconstrained end-effector under obstacle environment. Adv Robot. 2014;28:533–544.

[9] Kim J, Ko I, Park FC. Randomized path planning on foliated configuration spaces. In: Proceedings of KROS ubiquitous robots and ambient intelligence; Kuala Lumpur, Malaysia; 2014. p. 209–214.

[10] LaValle S. Rapidly-exploring random trees a new tool for path planning. Report No.: TR 98–11. Computer Science Department, Iowa State University.

[11] Siciliano B, Khatib O. Handbook of robotics. Berlin: Springer-Verlag; 2008.