공학석사 학위논문

불연속 다양체에서의 동작 계획

Motion Planning on Disconnected Manifolds: Manipulation under Task and Obstacle Constraints

2013년 2월

서울대학교 대학원

기계항공공학부

김 진 규

ABSTRACT

Motion Planning on Disconnected Manifolds: Manipulation under Task and Obstacle Constraints

by

Jinkyu Kim

School of Mechanical and Aerospace Engineering
Seoul National University

In this thesis we introduce an algorithm for motion planning on disconnected manifolds. Constraint manifolds can be disconnected if, for example, both task constraints and obstacles are present in the task space. Our proposed algorithm uses sampling-based motion planning on a foliation structure that consists of parallel submanifolds of lower dimension. Some case studies are performed to evaluate the performance of our proposed algorithm.

Keywords: Motion Planning, disconnected manifolds, foliation, rapidly-exploring random tree

Student Number: 2011-20698

Contents

List of Tables

List of Figures

1

Introduction

One of the most widely-used planners among sampling-based motion planning algorithms is the rapidly-exploring random tree (RRT) [1], [2]. Recently RRT-based algorithms have been developed to improve the planning performance by using some simple techniques [3], [4], [5] or to solve problems with various constraints [6], [7]. Generating more than two trees (basically two trees are built from start and goal configuration) in the configuration space [3] works well for the problem with narrow passages. Sampling constrained configurations [4] and considering path cost for growth of tree [5] can also improve the performance of motion planning. Constrained bi-direnctional rapidly-exploring random tree (CBiRRT) [6] and tangent space rapidly-exploring random tree (TSRRT) [7] can handle many kinds of constraints such as pose constraints, static torque limits, loop closure conditions, *etc*. What makes the CBiRRT able to handle those constraints is the projection of configurations to constraint manifolds by iterative method. TSRRT, however, reduces time of projection by using tangent space as an approximation of constraint manifolds.

Oriolo *et al* proposed a control-based algorithm for task-constrained motion [8]. The advantage of [8] compared to other task-constrained motion planners is the continued satisfaction of the constraint. The planner used motion generation scheme of differential model of the robot, so it can handle velocity and acceleration bounds. Furthermore, it used the foliation structure. The foliation structure consists of its parallel submanifolds of lower dimension and the planner searches constraint manifolds by a submanifold. The foliation structure is useful when the robot has kinematic redundancy.

A planner has been developed to solve a special problem with both task constraints and obstacles in the task space [9]. Constraint manifolds is possibly disconnected if both task constraints and obstacles exist in the task space at the same time. Releasing and re-grasping motion is needed to leap the disconnected region. Two kinds of planners are proposed in [9], $\mathbb{C}$-based planner and $\mathbb{T}$-based planner. $\mathbb{C}$-based planner makes two types of tree, the first one extends on constraint manifolds to make a path and the other one extends in the ambient configuration space to connect two trees of the former type. $\mathbb{T}$-based planner has a tree on the task constraints and each node has a configuration set in the configuration space. It finds a path between two configuration sets of two nearest nodes. $\mathbb{T}$-based planner outperforms $\mathbb{C}$-based planner because $\mathbb{T}$-based planner has much less projection procedure.

In this thesis we present a new algorithm for motion planning problems on disconnected manifolds. Our proposed planner solves this kind of problems by using the concept of foliation. We will call a submanifold in the foliation structure as a leaf. Our planner also has a tree on the task constraints and it is extended similar to CBiRRT algorithm. Every node of the tree has its own leaf because of redundancy. Our planner does not make a configuration set at a leaf, but has a single configuration and searches its surroundings with some conditions. Not making a

configuration set reduces the time to solve the problem because it takes less time for projection. Our planner also can make a natural motion because it uses very close configuration to proceed.

We list the concepts of some preliminaries including a foliation and explain the algorithm of our planner in Chapter 2. In Chapter 3, three case studies are performed and performance indices are reported compared to $\mathbb{T}$-based planner [9]. Finally, conclusion follows in Chapter 4.

2

Planning on Disconnected Manifolds

2.1 Preliminaries

This section contains some necessary concepts of motion planning problem, such as task space, configuration space and constraints. Moreover, a simple description of a foliation in motion planning problem is included.

2.1.1 Task Space and Configuration Space

Motion planning problem is to find a joint trajectory from given initial and final joint values or other given conditions. Let $\mathbb{T}$ denote the task space where a manipulator moves and constraints exist, and let $\mathbb{C}$ denote the configuration space also called joint space that consists of axes with each joint values of the manipulator.
We will use $\mathbf{x} \in \mathbb{T}$ as the position of the end-effector and $\mathbf{q} \in \mathbb{C}$ as its joint variables,

also called configuration. Forward kinematics can compute the position of the end-effector from given joint values by using kinematic equation.

$$\mathbf{x} = f(\mathbf{q}) \tag{2.1.1}$$

To compute configurations from the position of the end-effector is called inverse kinematics, and its equation is given by

$$\mathbf{q} = f^{-1}(\mathbf{x}) \tag{2.1.2}$$

Kinematic redundancy occurs if the number of degrees of freedom exceeds the dimension of the task space. Inverse kinematic solutions is not unique when the redundancy exists

2.1.2 Task Constraints

In this thesis, task constraints can be a constrained path, a surface or a higher dimensional structure. Task constraints are applied to the object not the end-effector. The end-effector should drive the object while satisfying task constraints and is allowed to release and re-grasp the object if needed. Task constraints can be expressed in

$$\mathbf{t}_c(\mathbf{x}) = 0 \quad \text{where} \quad \mathbf{x} \in \mathbb{T} \tag{2.1.3}$$

We can divide the task space into two subspaces, space satisfying task constrains and not. Let $C_{\mathbb{T}} \in \mathbb{T}$ denote the former one and the latter one is the complementary set of $C_{\mathbb{T}}$ in $\mathbb{T}$.

We can also divide the configuration space into two subspaces by using Equation (2.1.1). Let $C_{\mathbb{C}} \in \mathbb{C}$ denote the constraint manifolds in $\mathbb{C}$ and it satisfies:

$$C_{\mathbb{C}} = \{\mathbf{q} \mid \mathbf{t}_c(f(\mathbf{q})) = 0\} \quad \text{where } \mathbf{q} \in \mathbb{C} \tag{2.1.4}$$

2.1.3 Foliation

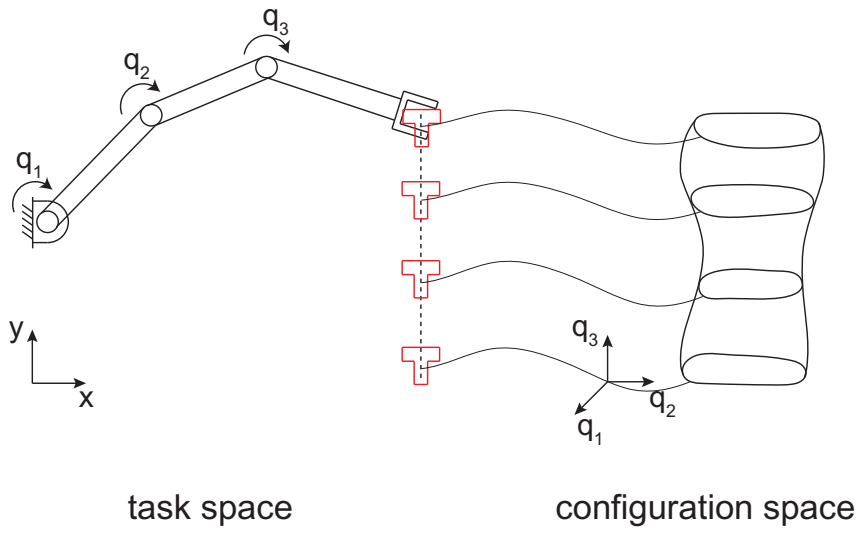
A motivational example is sketched in Figure 2.1. 3-DOF planar manipulator, T-shape object and one circular obstacle are placed in the task space and constraint manifolds $C_{\mathbb{C}}$ are described in the configuration space. The object must lie on the dotted line, task path constraint.

The manipulator has the redundancy because its DOF is larger than the task space dimension. Various poses of the manipulator can be found to grasp the object placed at specific position. It is described in configuration space as a ring-shape closed curve. We will call this curve a leaf. In other words, all configurations at the same leaf have the same end-effector position.

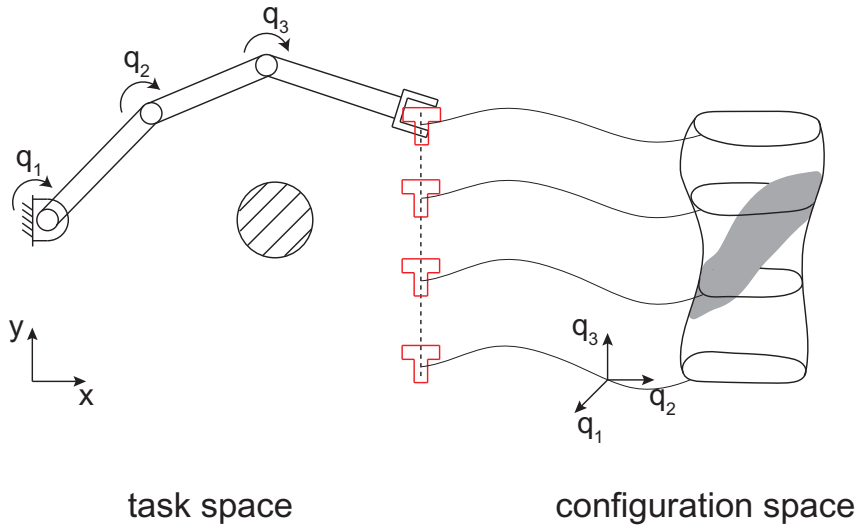
As can be seen in the Figure 2.1, constraint manifolds $C_{\mathbb{C}}$ consist of many parallel submanifolds of lower dimension. This structure is called a foliation and more mathematical description is given in the book [10].

2.2 Problem Definition

Figure 2.1 describes the motivational example. Our goal is to move the object following the task path constraint from top to bottom while avoiding collision with the obstacle. The solution trajectory of joint values must lie on $C_{\mathbb{C}}$. Infeasible region is formed in constraint manifolds, shaded region, because of the obstacle in the task space (see Figure 2.1(b)). These are called disconnected manifolds that we are dealing with in this thesis. To leap shaded region, the manipulator should release the object, move to a different pose than before and re-grasp the object.



(a) A problem without obstacles



(b) A problem with an obstacle

Figure 2.1: A motivational Example

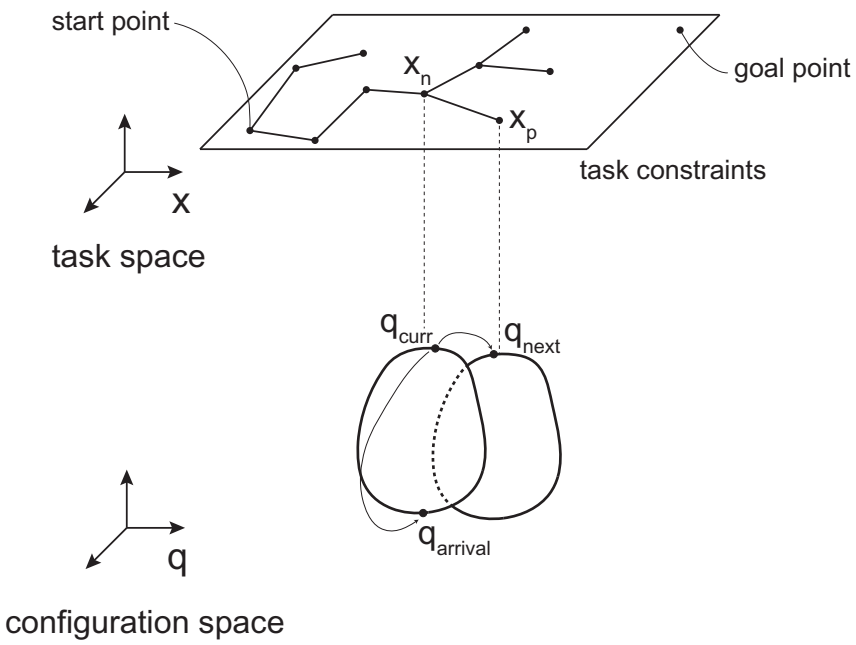


Figure 2.2: A simple illustration of the proposed planner

To make problem more general, task constraints, from Equation (2.1.3), can be a surface or a higher dimensional structure seen in Figure 2.2. The planning problem with two or more disconnected manifolds due to multiple obstacles is our goal.

2.3 Proposed Algorithm

Our proposed planner has a tree, $T_{\mathbb{T}}$, in the task space and every node of the tree has its own leaf. In Figure 2.2, $\mathbf{x}_{\mathrm{p}}$ is the node may be added to the tree and $\mathbf{x}_{\mathrm{n}}$ is the nearest node of $\mathbf{x}_{\mathrm{p}}$ in $T_{\mathbb{T}}$. Leaves of two nodes are described in the figure. The planner solves the problem by a leaf. First, the tree is extended in the task space and load the saved configuration at the leaf of original node, $\mathbf{q}_{\mathrm{curr}}$. Then, the planner decides whether to proceed to the next leaf or to jump to another configuration at

the same leaf.

Whole algorithm of the planner is described in Table 2.1.

	Whole Algorithm
1:	Tree $T_{\mathbb{T}}$.initialize with start position
2:	**repeat** {
3:	Extension($T_{\mathbb{T}}$)
4:	**if** distance($\mathbf{x}_{\mathrm{p}}$, goal position) $<$ stepsize
5:	$\mathbf{x}_{\mathrm{p}} \leftarrow$ goal position
6:	bool CloseToGoal = true
7:	**end if**
8:	**switch** (Decision($\mathbf{q}_{\mathrm{curr}}$, $\mathbf{x}_{\mathrm{n}}$, $\mathbf{x}_{\mathrm{p}}$))
9:	**case** 1: ProceedToNextLeaf
10:	**case** 2: JumpInSameLeaf
11:	**end switch**
12:	} **until** CloseToGoal = true && case 1 is performed
13:	**return** Success

Table 2.1: Whole Algorithm

At first line, '$T_{\mathbb{T}}$.initialize' means making a tree in $\mathbb{T}$ and setting root node with start position in $\mathbb{T}$ and a configuration at its leaf. The functions of Extension($T_{\mathbb{T}}$), Decision($\mathbf{q}_{\mathrm{curr}}$), ProceedToNextLeaf and JumpInSameLeaf will be explained in following subsections. The variable stepsize at line 4 is the pre-defined value and also used in Extension function. The function 'distance' at the same line calculates Euclidean distance between two configurations.

2.3.1 Extension in Task Space

The tree, $T_{\mathbb{T}}$, searches whole task space like a tree in RRT algorithm [2]. It is more similar with CBi-RRT algorithm [6] because we deal with task constraints problem. Table 2.2 shows this algorithm.

	Extension($T_{\mathbb{T}}$)
1:	**repeat** {
2:	**if** random(0.0, 1.0) > probability of goal
3:	$\mathbf{x}_{\mathrm{r}} \leftarrow$ Random($\mathbb{T}$)
4:	**else**
5:	$\mathbf{x}_{\mathrm{r}} \leftarrow$ goal position
6:	**end if**
7:	$\mathbf{x}_{\mathrm{n}} \leftarrow$ NearestNeighbor($T_{\mathbb{T}}, \mathbf{x}_{\mathrm{r}}$)
8:	$\mathbf{x}_{\mathrm{e}} \leftarrow$ Extend($\mathbf{x}_{\mathrm{n}}, \mathbf{x}_{\mathrm{r}}$, stepsize)
9:	$\mathbf{x}_{\mathrm{p}} \leftarrow$ ProjectToT($\mathbf{x}_{\mathrm{e}}, C_{\mathbb{T}}$)
10:	} **until** $\mathbf{x}_{\mathrm{p}}$ does not collide with obstacles
11:	add $\mathbf{x}_{\mathrm{p}}$ to $T_{\mathbb{T}}$
12:	**return** $T_{\mathbb{T}}$

Table 2.2: RRT-like extension in the task space

From line 2 to 6 describes the procedure that we use the goal position as a random node in the chance of pre-defined probability. The function NearestNeighbor($T_{\mathbb{T}}, \mathbf{x}_{\mathrm{r}}$) finds a nearest node in the tree $T_{\mathbb{T}}$ from random node $\mathbf{x}_{\mathrm{r}}$. In Extend($\mathbf{x}_{\mathrm{n}}, \mathbf{x}_{\mathrm{r}}$, stepsize) function, $\mathbf{x}_{\mathrm{n}}$ is extended toward $\mathbf{x}_{\mathrm{r}}$ as far as pre-defined stepsize. ProjectToT($\mathbf{x}_{\mathrm{e}}, C_{\mathbb{T}}$) function projects $\mathbf{x}_{\mathrm{e}}$ to the task constraints, $C_{\mathbb{T}}$, by using Newton-Raphson method (will be discussed in Section 2.3.3).

2.3.2 Decision Whether to Proceed or to Jump

This section explains some conditions to decide whether to proceed to the next leaf or to jump to another configuration at the same leaf. After decision, the planner finds a path from $\mathbf{q}_{\text{curr}}$ to $\mathbf{q}_{\text{next}}$ or from $\mathbf{q}_{\text{curr}}$ to $\mathbf{q}_{\text{arrival}}$.

Given: $\mathbf{q}_{\text{curr}}$, $\mathbf{x}_{\text{n}}$, $\mathbf{x}_{\text{p}}$ from Extension function

	Decision($\mathbf{q}_{\text{curr}}$, $\mathbf{x}_{\text{n}}$, $\mathbf{x}_{\text{p}}$)
1:	$\mathbf{q}_{\text{next}} \leftarrow$ ProjectToLeaf($\mathbf{q}_{\text{curr}}, \mathbf{x}_{\text{p}}$)
2:	**if** $\mathbf{q}_{\text{next}}$ collides with obstacles
3:	**repeat** {
4:	$\mathbf{q}_{\text{next}} \leftarrow$ ProjectToLeaf($\mathbf{q}_{\text{rand}}, \mathbf{x}_{\text{p}}$)
5:	} **until** $\mathbf{q}_{\text{next}}$ does not collide with obstacles
6:	**end if**
7:	$\overline{\mathbf{q}_{\text{curr}}\mathbf{q}_{\text{next}}} \leftarrow$ LineSegment($\mathbf{q}_{\text{curr}}, \mathbf{q}_{\text{next}}$)
8:	**if** the path $\overline{\mathbf{q}_{\text{curr}}\mathbf{q}_{\text{next}}}$ does not pass via obstacle region
9:	**return** ProceedToNextLeaf
10:	**else**
11:	**return** JumpInSameLeaf
12:	**end if**

Table 2.3: Decision whether to proceed or to jump

The function ProjectToLeaf($\mathbf{q}, \mathbf{x}$) projects $\mathbf{q}$ to $\mathbf{x}$ and will be discussed in Section 2.3.3. In Section 2.3.4, LineSegment function will be explained that finds a path on $C_{\mathbb{C}}$. At line 8, 'passing via obstacle region of a path' means that one or more configurations in the path collide with obstacles. Lastly, $\mathbf{q}_{\text{rand}}$ is a random configuration in $\mathbb{C}$ not exceeding joint limits of the manipulator.

ProceedToNextLeaf	
1:	save the path $\overline{\mathbf{q}_{\mathrm{curr}}\mathbf{q}_{\mathrm{next}}}$ to JointTrajectory

Table 2.4: Proceeding to next leaf

The path $\overline{\mathbf{q}_{\mathrm{curr}}\mathbf{q}_{\mathrm{next}}}$ is given from Table 2.3 and JointTrajectory is a desired solution of whole algorithm.

JumpInSameLeaf	
1:	**repeat** {
2:	$\mathbf{q}_{\mathrm{arrival}} \leftarrow \mathrm{ProjectToLeaf}(\mathbf{q}_{\mathrm{rand}}, \mathbf{x}_{\mathrm{n}})$
3:	$\overline{\mathbf{q}_{\mathrm{curr}}\mathbf{q}_{\mathrm{arrival}}} \leftarrow \mathrm{LineSegment}\ (\mathbf{q}_{\mathrm{curr}}, \mathbf{q}_{\mathrm{arrival}})$
4:	} **until** $\mathbf{q}_{\mathrm{arrival}}$ does not collide && $\overline{\mathbf{q}_{\mathrm{curr}}\mathbf{q}_{\mathrm{arrival}}}$ pass via obstacle region
5:	$\overline{\mathbf{q}_{\mathrm{curr}}\mathbf{q}_{\mathrm{arrival}}} \leftarrow \mathrm{BasicRRT}\ (\mathbf{q}_{\mathrm{curr}}, \mathbf{q}_{\mathrm{arrival}})$
6:	save the path $\overline{\mathbf{q}_{\mathrm{curr}}\mathbf{q}_{\mathrm{arrival}}}$ to JointTrajectory

Table 2.5: Jumping in same leaf

The motion of the manipulator in the algorithm of Table 2.5 is release the object, moving to a different pose than before and re-grasp the object. Intuitively, the condition '$\overline{\mathbf{q}_{\mathrm{curr}}\mathbf{q}_{\mathrm{arrival}}}$ pass via obstacle region' is needed to leap obstacles that make constraint manifolds disconnected. At line 5, BasicRRT function finds a path from $\mathbf{q}_{\mathrm{curr}}$ to $\mathbf{q}_{\mathrm{arrival}}$ by using RRT algorithm [2].

2.3.3 Projection

We have three kinds of projections, ProjectToT, ProjectToC and ProjectToLeaf. $\mathrm{ProjectToT}(\mathbf{x}, C_{\mathbb{T}})$ projects $\mathbf{x}$ to task constraints in $\mathbb{T}$ and $\mathrm{ProjectToC}(\mathbf{q}, C_{\mathbb{C}})$ projects a configuration $\mathbf{q}$ to constraint manifolds in $\mathbb{C}$. $\mathrm{ProjectToLeaf}(\mathbf{q}, \mathbf{x})$ is the inverse

kinematics and it returns a configuration $\mathbf{q}_{\text{ret}}$ that satisfying $f(\mathbf{q}_{\text{ret}}) = \mathbf{x}$ with $\mathbf{q}$ as an initial guess. We used Newton-Raphson method, which needs to define error function $e = g(\mathbf{x})$ or $e = g(\mathbf{q})$, for projection. The algorithm in Table 2.6 is written with $e = g(\mathbf{x})$.

ProjectToT, ProjectToC or ProjectToLeaf	
1:	$e \leftarrow g(\mathbf{x})$
2:	**repeat** {
3:	$\mathbf{x} \leftarrow \mathbf{x} - J(\mathbf{x})^{\dagger} e$
4:	$e \leftarrow g(\mathbf{x})$
5:	} **until** $\parallel e \parallel < \epsilon$
6:	**return x**

Table 2.6: Projection by Newton-Raphson method

To calculate $J(\mathbf{x})^{\dagger}$, use the first order Tayler expansion of $g(\mathbf{x})$ first of all.

$$e + \delta e \simeq g(\mathbf{x}) + \frac{\partial g}{\partial \mathbf{x}}(\mathbf{x})\delta\mathbf{x} \tag{2.3.5}$$

We denote the Jacobian matrix $\frac{\partial g}{\partial \mathbf{x}}(\mathbf{x})$ by $J(\mathbf{x})$. Solving for $\delta\mathbf{x}$, we get the update rule

$$\mathbf{x}_{\text{new}} \leftarrow \mathbf{x} - J(\mathbf{x})^{\dagger}\delta e \tag{2.3.6}$$

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

$$J(\mathbf{x})^{\dagger} = J(\mathbf{x})^{T}[J(\mathbf{x})J(\mathbf{x})^{T}]^{-1} \tag{2.3.7}$$

Lastly, error function is defined as

- ProjectToT: $e(\mathbf{x}) = \text{norm}(\mathbf{x}, C_{\mathbb{T}})$
- ProjectToC: $e(\mathbf{q}) = \text{norm}(f(\mathbf{q}), C_{\mathbb{T}})$
- ProjectToLeaf: $e(\mathbf{q}) = \text{norm}(f(\mathbf{q}), \mathbf{x})$

We used Euclidean norm for ProjectToLeaf function and defined norm operation properly to the task constraints for others.

2.3.4 Line Segment

The function LineSegment($\mathbf{q}_1, \mathbf{q}_2$) returns a path on $C_{\mathbb{C}}$ from ProjectToC($\mathbf{q}_1, C_{\mathbb{C}}$) to ProjectToC($\mathbf{q}_2, C_{\mathbb{C}}$).

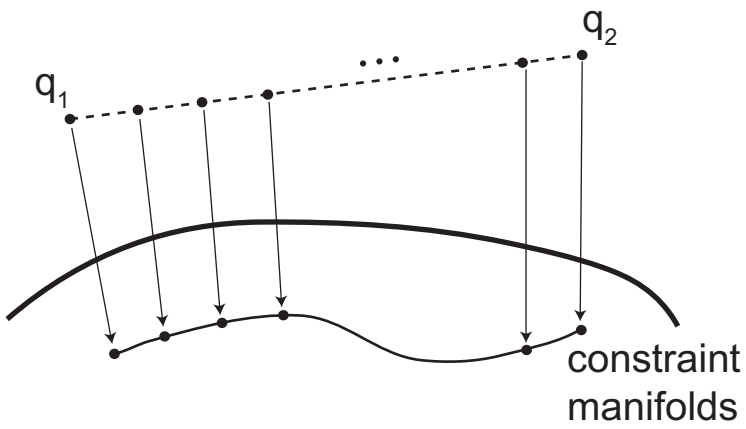


Figure 2.3: Explanation of LineSegment function

This function works as follows:

- Draw a linear line from $\mathbf{q}_1$ to $\mathbf{q}_2$.
- Chop the line into several pieces in pre-decided size.
- Project every piece to constraint manifolds by using function ProjectToC($\mathbf{q}, C_{\mathbb{C}}$).
- Return projected trajectory.

3

Case Studies

We performed three simulation of case studies to evaluate the performance of the proposed algorithm. All simulations are performed with Intel Core2 Quad processor, Q9450, @ 2.66GHz and 5.0GB RAM on Windows 7. We compared operation time, the number of projection, the number of leap and the path length in the configuration space to the algorithm of $\mathbb{T}$-based MMP-UE method in [9]. Projection is counted when the functions ProjectToLeaf and ProjectToC are called (Section 2.3.3). Let Q denote the solution path of the problem, then the path length, l, is defined as

$$l = \sum_{j}^{m-1} \sum_{i}^{n} \mid Q(i, j+1) - Q(i, j) \mid \tag{3.0.1}$$

where $Q(i, j)$ is the i-th component of j-th element in Q (i is the variable for the configuration space and j for the number of configurations in Q), n is the dimension of the configuration space and m is the number of configurations in Q. The path length can represent how much the joint values of the manipulator changes and how natural the result motion is. All performance indices are calculated from the solution

path Q with omitted leap path because the principal purpose of the planner is to find where or when to leap. Furthermore, the number of leap varies to solutions. So comparing operation time and path length with whole configurations, including leap path, is meaningless. Reported values of the performance indices are the average of 20 times of trial.

In [9], a set of configurations is needed for each node of the tree in the task space. The number of configurations in one set, n_c, is an important and influential variable to the results. So, we performed simulations several values of n_c and compared all of them.

3.1 3 DOF Planar Manipulator

This simulation is performed in MATLAB and 2 circular obstacles and linear path constraint are placed in the task space. The length of all links are 1, obstacles are placed at (1.5, 0.85) and (1.5, -0.85) with their radius 0.1 and the line from (2.0, 2.0) to (2.0, -2.0) is the path constraint. We extend the tree in the task space to the goal node (set the probability of goal as 1 in Table 2.2) because the task constraint is a path.

Simulation results are expressed in Table 3.1 and Figure 3.1. Our planner outperform $\mathbb{T}$-based planner [9] in all of the performance indices. $\mathbb{T}$-based planner takes much more time than ours because the number of projection is higher. The path length gets lower as n_c goes higher among $\mathbb{T}$-based planner. The motion is shown in Figure 3.1, and the manipulator release the object at (c) and re-grasp at (g). It is also found at (i) and (m).

3.2 6 DOF Spatial Manipulator 1

This simulation is performed in C++ language and used MPNN library [11] to find a nearest neighbor node. Linear path constraint and 2 pillar-like obstacles are placed in the task space. First link's length is 0.5 and the others' lengths are 0.35. Two pillar-like obstacles are located at (0.35, 0.25, 0.0) and (0.35, -0.25, 0.0). The path constraint is given from (1.0, 0.7, 0.0) to (1.0, -0.7, 0.0). We set the probability of goal as 1 because of the same reason in case 1.

Simulation results are expressed in Table 3.2 and Figure 3.2. Our planner takes much less time due to smaller number of projection. It is also reported that our planner's number of leap and the path length are less than $\mathbb{T}$-based planner. It means that the solution path from our planner is more natural. The motion without leap is shown in Figure 3.2. The manipulator release the object and re-grasp it at (f)-(g) and (k)-(l).

3.3 6 DOF Spatial Manipulator 2

This simulation is also performed in C++ language, used MPNN library [11] and has the same manipulator with case 2. The environment of this case is shown in Figure 3.3. Start and goal position are shown in Figure 3.3(a) and (b). The big flat surface is the task constraint where the object must lie on and the smaller object is the obstacle which is placed above the surface (see Figure 3.3(c) and (d)). In this case, the manipulator cannot deliver the object along the linear line from the start to the goal position because the obstacle blocked it. So, instead of extending the tree to the goal position, it should search the task constraint region randomly. The probability of goal is set as 0.2.

The results are shown in Table 3.3 and Figure 3.4. Our planner takes less time and has the lower number of projection and path length. The leap motion, however, is too much because the environment is complicated and our planner decides to jump whenever the obstacle is detected.

			Time (sec)	# of projection	# of leap	Path length
Our algorithm			0.62	811.85	2.50	5.54
$\mathbb{T}$-based planner	n_c	3	34.13	7414.55	5.40	46.95
		5	13.04	4507.00	3.80	30.50
		10	3.27	3265.90	2.90	21.24
		20	2.69	2503.10	2.60	12.07
		50	3.94	2980.00	3.10	9.44
		100	7.14	4716.00	2.80	8.89

Table 3.1: Simulation results of 3 DOF planar manipulator

			Time (sec)	# of projection	# of leap	Path length
Our algorithm			9.00	987.55	2.25	8.11
$\mathbb{T}$-based planner	n_c	3	101.26	9848.05	8.75	112.23
		5	85.73	9619.40	5.70	94.34
		10	94.77	11892.65	4.35	72.00
		20	150.39	18984.95	3.25	57.83
		50	328.26	41600.90	2.50	42.19

Table 3.2: Simulation results of 6 DOF spatial manipulator (1)

			Time (sec)	# of projection	# of leap	Path length
Our algorithm			102.93	15175.75	10.25	8.03
$\mathbb{T}$-based planner	n_c	3	1340.77	191746.15	23.05	164.44
		5	432.09	65518.30	26.25	135.50
		10	676.35	99554.25	21.00	130.74
		20	1345.04	186565.85	7.20	122.01
		50	1598.12	234124.05	3.60	78.14

Table 3.3: Simulation results of 6 DOF spatial manipulator (2)

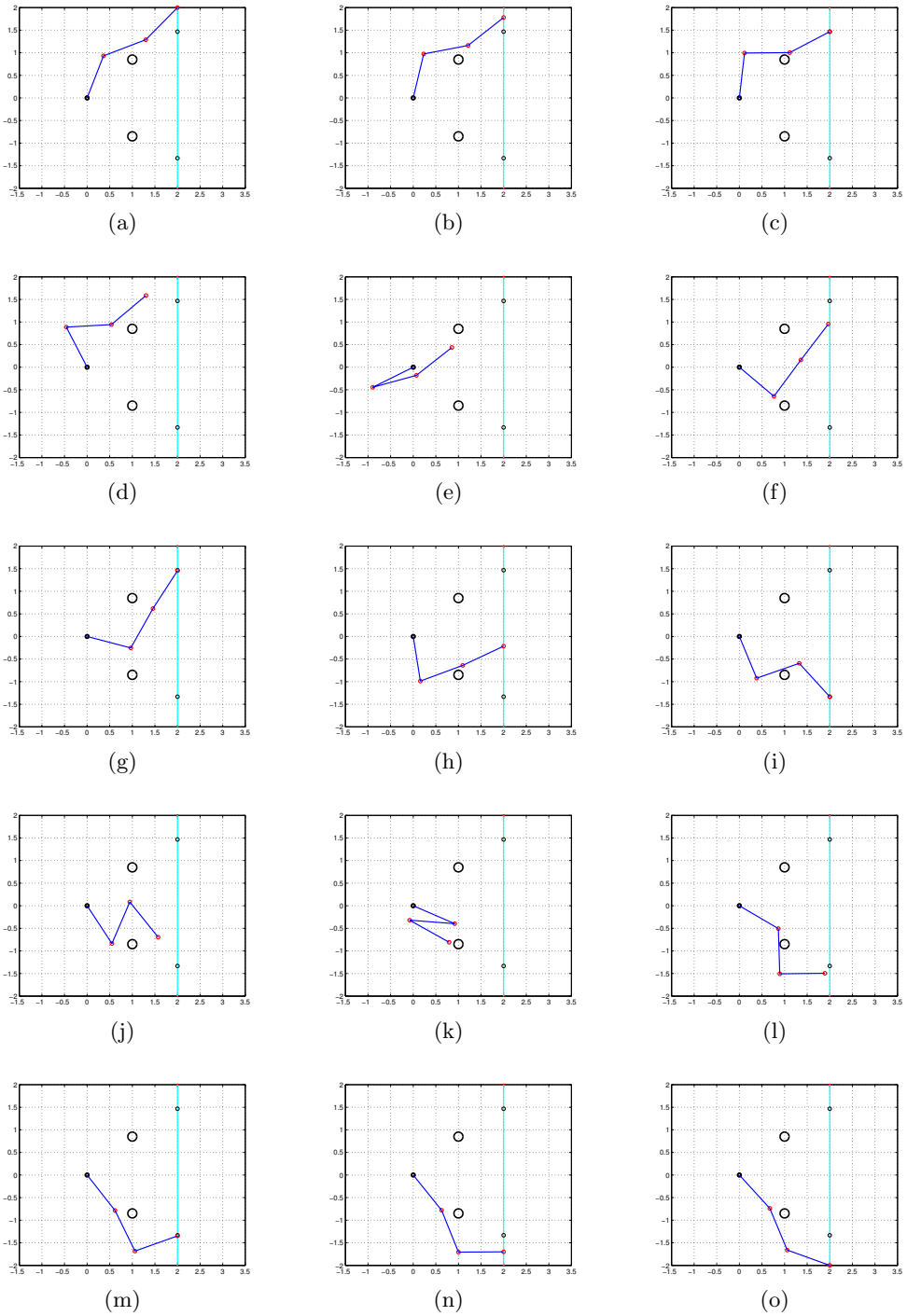


Figure 3.1: Simulation results of 3 DOF planar manipulator

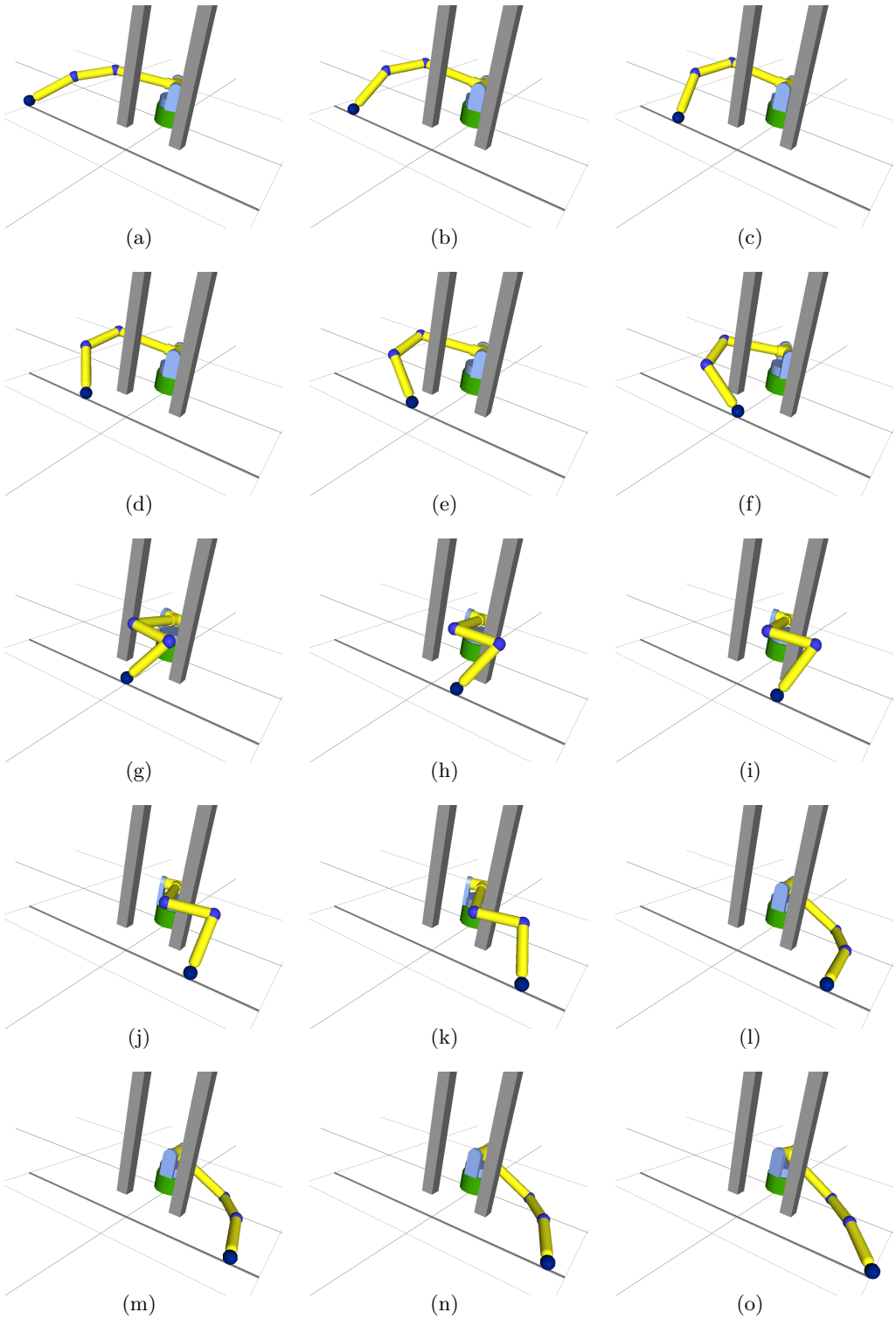
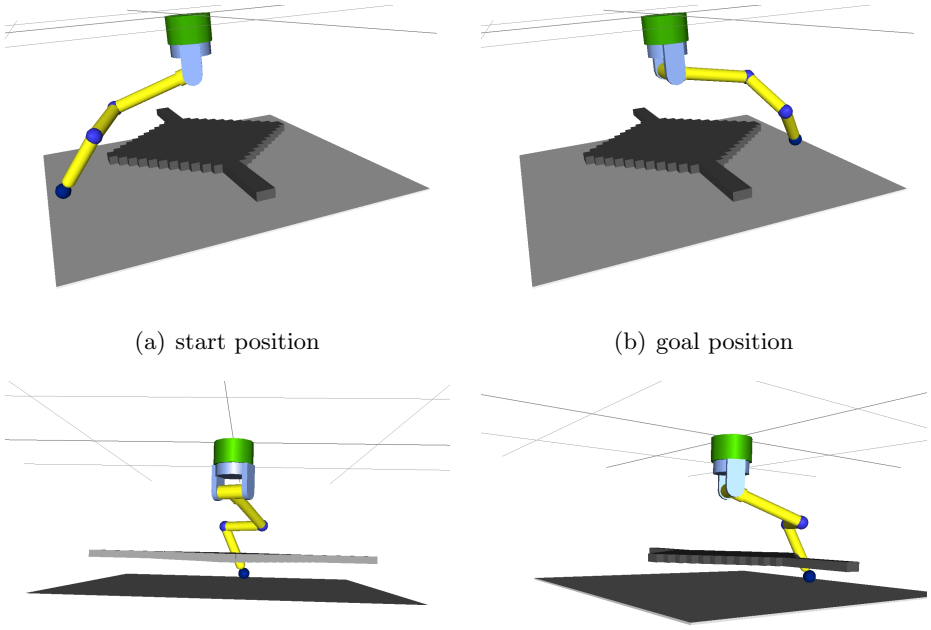


Figure 3.2: Simulation results of 6 DOF spatial manipulator (1)



(a) start position

(b) goal position

(c) the obstacle and the surface constraint

(d) the obstacle and the surface constraint

Figure 3.3: Problem definition of 6 DOF spatial manipulator (2)

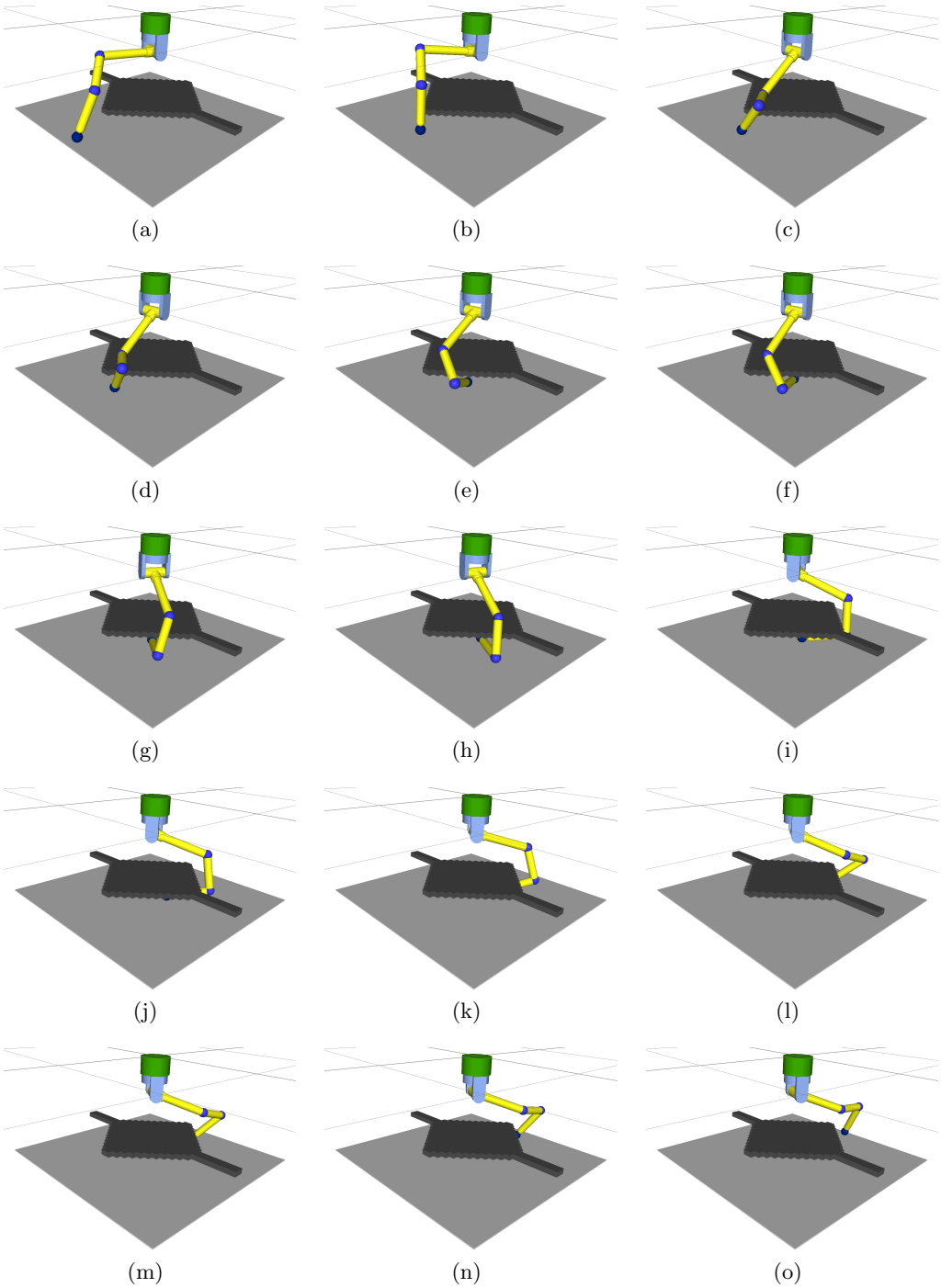


Figure 3.4: Simulation results of 6 DOF spatial manipulator (2)

4

Conclusion

This thesis has presented a new algorithm for motion planning on disconnected manifolds. Task constraints and obstacles can make constraint manifolds disconnected, and the solution must leap the obstacles. The key point of solving this kind of problems is to find where or when to leap.

The distinct feature between our algorithm and other existing algorithms is that our planner uses the concept of a foliation structure. A foliation structure consists of lower dimensional submanifolds, called leaves. Our planner has only one configuration at a leaf, instead of making a set of configurations. Planning based on the leaves makes our planner outperform other algorithms. It also has the features of RRT, sampling-based motion planning. It has a tree on the task constraints to deal with higher dimensional task constraints problem and uses a basic RRT algorithm to find a path of releasing and re-grasping motion.

A problem with multiple obstacles that divide constraint manifolds into several pieces or higher dimension of the configuration space (simulated up to 6) can be solved by our planner. This planner can be applied to real robots of sewing or

moving a toy train through the tunnel.

The results which are shown in Section 3 evaluate that our planner takes much less time, has smaller number of projections and leap. The path length, however, depends on cases.

As can be seen from the results the number of leap is more than ideal number, especially in case 3. Usually, ideal number equals to the number of obstacles. It comes from that the planner does not know how the constraint manifolds look like and how many times leap is needed. It only knows where or when to release and re-grasp the object. It is being investigated now and could be resolved by post-processing.

Bibliography

[1] S.M. LaValle. Rapidly-exploring random trees a new tool for path planning. 1998.

[2] S.M. LaValle and J.J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.

[3] M. Strandberg. Augmenting rrt-planners with local trees. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3258–3262. IEEE, 2004.

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

[5] C. Urmson and R. Simmons. Approaches for heuristically biasing rrt growth. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1178–1183. IEEE, 2003.

[6] D. Berenson, S.S. Srinivasa, D. Ferguson, and J.J. Kuffner. Manipulation planning on constraint manifolds. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 625–632. IEEE, 2009.

[7] C. Suh, T.T. Um, B. Kim, H. Noh, M. Kim, and F.C. Park. Tangent space rrt: A randomized planning algorithm on constraint manifolds. In *Robotics*

and Automation (ICRA), 2011 IEEE International Conference on, pages 4968–4973. IEEE, 2011.

[8] G. Oriolo and M. Vendittelli. A control-based approach to task-constrained motion planning. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 297–302. IEEE, 2009.

[9] Jung-Tae. Kim. *Motion Planning for Manipulator under End-effector and Obstacle Constraint.* PhD thesis, Computer Science and Engineering, Pohang Unversity of Science and Technology, 2012.

[10] D. Rolfsen. *Knots and Links.* American Mathematical Society, 1976.

[11] A. Yershova and S.M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *Robotics, IEEE Transactions on*, 23(1):151–157, 2007.

국문초록

본 논문에서는 끊어진 다양체에서의 동작 계획의 새로운 알고리즘을 제안하였다. 제한 조건을 만족하는 관절 공간 내의 다양체는 작업공간에 제한 조건과 장애물이 동시에 존재하는 경우 끊어질 수도 있다. 제안하는 알고리즘은 foliation(기하학, 엽리구조)상에서 rapidly-exploring random tree(확률 기반의 동작 계획)를 이용한다. Foliation은 그보다 작은 차원의 평행한 하위 다양체들로 구성되어 있으며 하위 다양체를 단위로 하여 관절 경로를 찾는 것이 제안하는 알고리즘의 핵심 기능이다. 몇 가지의 예제를 통하여 제안하는 알고리즘의 성능을 기존의 다른 알고리즘과 비교하여 검증한다.

주요어: 동작 계획, 불연속 다양체, 확률 기반

학번: 2011-20698