박사 학위논문
Ph. D. Dissertation

좁은 길 문제를 해결하기 위한 샘플링 기반 모션 플래닝 알고리즘

Sampling-based motion planning algorithm
to handle a narrow passage problem

이 정 환 (李 政 奐 Lee, Junghwan)
전산학과
Department of Computer Science

KAIST

2014

좁은 길 문제를 해결하기 위한 샘플링 기반 모션 플래닝 알고리즘

Sampling-based motion planning algorithm
to handle a narrow passage problem

Sampling-based motion planning algorithm to handle a narrow passage problem

Advisor : Professor Yoon, Sung-Eui

by

Lee, Junghwan

Department of Computer Science

KAIST

A thesis submitted to the faculty of KAIST in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science . The study was conducted in accordance with Code of Research Ethics[1].

2014. 5. 12.

Approved by

Professor Yoon, Sung-Eui

[Advisor]

[1]Declaration of Ethical Conduct in Research: I, as a graduate student of KAIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

좁은 길 문제를 해결하기 위한 샘플링 기반 모션 플래닝 알고리즘

이 정 환

위 논문은 한국과학기술원 박사학위논문으로
학위논문심사위원회에서 심사 통과하였음.

2014년 5월 12일

DCS 20097002

이 정 환. Lee, Junghwan. Sampling-based motion planning algorithm to handle a narrow passage problem. 좁은 길 문제를 해결하기 위한 샘플링 기반 모션 플래닝 알고리즘. Department of Computer Science . 2014. 53p. Advisor Prof. Yoon, Sung-Eui. Text in English.

ABSTRACT

Robot motion planning problem has been actively studied since 1970's and has many applications, such as part disassembly, autonomous vehicle and computer graphics. Sampling-based algorithms have been successfully used to give a *probabilistically complete* solution for a variety of types of robots and constraints. Sampling-based algorithms, however, suffer when narrow passages are included in an environment. This phenomenon become worse as more constraints are related such as kinematic and dynamics constraints, because of its high degrees-of-freedom and complexity of extensions. In this thesis, we present novel sampling-based planners to efficiently handle various environments that have different characteristics and constraints. We first present a bridge line-test that can identify narrow passage regions in the configuration space, and then selectively performs an optimization-based retraction only at those regions. We also propose a non-colliding line-test, a dual operator to the bridge line-test, as a culling method to avoid generating samples near wide-open free spaces. These two line-tests are performed with a small computational overhead. As a result, proposed planner shows better performance than recent works for a variety of environments that have or do not have narrow passages. For a hyper-redundant robot manipulator, we define productive regions in the task space as a set of states that can lead effectively to a goal state. To check whether a node of a random tree is in productive regions or not, we construct a maximum reachable area (MRA) for a node in the task space, where a manipulator can reach from the node by using an employed local planner. When the MRA of a node contains the goal state, we call it *promising* and bias our sampling to cover promising MRAs. When the MRA does not contain the goal state, we call it *unpromising* and construct a detour sampling domain for detouring operations from obstacles constraining the manipulator. The union of promising MRAs and detour sampling domains approximates our productive regions, and we bias sampling regions to cover these domains more. Finally, we present a novel kinodynamic motion planner designed for complex dynamics environments with many obstacles. We define a motion database that is constructed as a preprocessing and can replace time-consuming propagation function integration of kinodynamic sampling-based planners. Motion interpolation is applied for retrieving large set of motions in the continuous state space. During iterations, interpolated motions are selectively validated by the employed simulator in order to increase the efficiency of the planner. Constructed motion database can be also used for other problems such as different composition of obstacles or different start/goal state. Overall, proposed methods achieve several times improvement over the state-of-the-art planning algorithms.

Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Sampling-based Motion Planning

Robot motion planning problem is computing collision-free paths of a robot from given initial state to a goal state or region, while satisfying a set of constraints. The problem has been actively studied since 1970's and has many applications, such as part disassembly simulation, autonomous vehicle, tolerance verification, protein folding, and computer graphics [CL95, FK99, Lat99]. Reif proved that a most basic version of motion planning problems, such as a piano movers' problem, is PSPACE-hard to develop a complete solution [Rei79].

Sampling-based motion planning algorithms have been designed and successfully used to solve the problem for a variety of environments under the limited completeness which is *probabilistically complete.* Two of the most successful algorithms include Probabilistic Roadmap Method (PRM) [KSLO96] and Rapidly-exploring Random Tree (RRT) [KL00]. At a high level, as random sampling is performed iteratively, these techniques provide collision-free paths by capturing a larger portion of the connectivity of the free space.

One of the most challenging cases for sampling-based techniques is to efficiently handle narrow passages in the free space. In many practical motion planning applications, a robot often needs to pass through narrow passages and the performance of sampling-based planning algorithms can degrade significantly. Many approaches have been proposed in different directions such as utilizing the workspace geometric information [KH06], filtering (or adaptive sampling) techniques towards more important regions [BOvdS99, SLN00, HJRS03, YJSL05], retracting samples, etc.

Recently, retraction-based techniques [ZM08, RTLA06, HSAlCL06, RL05, SL05] have been actively studied, since they can pose samples near the boundary of the C-Obstacle, efficiently leading to explore important regions including narrow passages. However, these techniques have the computation overhead of retracting sampling near the boundary of the C-Obstacle and thus can run slower even to a basic RRT method, when the free space does not have such difficult regions [ZM08].

1.2 Motion Planning with Kinematic Constraints

The motion planning problem for robot manipulators is to generate a motion trajectory to move from an initial configuration to reach an end-effector in a goal position, while satisfying desired constraints (e.g. avoiding obstacles, joint limits, etc.). Specifically, avoiding obstacles is the principal constraint for kinematically redundant manipulators, which have high degrees-of-freedom (dofs). Recently, sampling-based path planners including the Rapidly-exploring Random Tree (RRT) have been successfully used to give a probabilistically complete solution for redundant manipulators. Most sampling-based planners work in the configuration space (or joint space) and experience difficulty as the number of dimensions of the configuration space grows.

This phenomenon becomes worse for hyper-redundant manipulators whose configuration space has much higher dimensions than common task spaces (e.g. three dimensions when a task is positioning an end-effector in a 3D workspace). Normally more than 10-dofs for rigid link manipulators are considered to be hyper-redundant. These extra redundant dofs give the manipulator greater dexterity and potential for maneuvering within tight environments. Therefore robots for disaster relief or medical surgery are representative applications for hyper-redundant manipulators, to name a few. One example of hyper-redundant manipulators is the 30-dofs robot by Chirikjian and Burdick [CB93].

Recently, a concept of utilizing the task space has received a strong attention for boosting the performance of planners. Some of them use a task space distance function [BKDA06] , exploit the task space as a goal bias [VWFS07], or perform direct task space exploration [ST09]. Specifically, directly exploring the task space can enable planners to effectively solve high dimensional problems by using a much lower dimensional task space.

1.3 Motion Planning with Kinodynamic Constraints

Planning for real world robots which move and interact with environments under the physical laws, requires another class of constraints, kinodynamic constraints [DXCR93], which includes non-holonomic constraints for underactuated systems and differential constraints for dynamical systems. The kinodynamic motion planning can provide valuable solution in a wide variety of applications such as space robots, mobile robots with wheels, humanoids, underwater robots, helicopters, etc.

There are two main approaches in robotics research for solving the kinodynamic planning problem. The first approach decouples the problem by solving basic path planning in robot's configuration space and then finding a solution path with proper controls that satisfies the dynamics and other constraints [SD91, KJKN$^+$02, DRF$^+$08a]. A solution found in the first step, considers only kinematics and geometric constraints while ignoring the dynamics systems, therefore the path would be impossible to track under kinodynamic constraints in the second step.

The second approach is a generalization of planning in configuration space by employing the state space that includes both configuration spaces components and its velocity components. By the generalization, many sampling-based motion planning techniques can be used for kinodynamic problem [LK01, HKLR02a] with a suitable metric and extension methods under dynamics systems, and resultant planner become probabilistically complete as the sampling-based planners are.

Chapter 2. Related Work

2.1 Sampling-based Motion Planning

The sampling-based algorithms have been successfully used to solve various motion planning problems in practice. Among them, Probabilistic Roadmap Method (PRM) [KSLO96] and Rapidly-exploring Random Tree (RRT) [KL00] are the most widely used approaches [LaV06]. These techniques have been extensively studied and an excellent survey is available [HLK06].

Specifically, RRT methods have been extended to solve a wide variety of practical single-query problems in robotics and other related domains. [BV02, FKS06, BCLM06, CJS07]. RRT methods have been improved in many different directions by considering workspace information [KH06, BB07], biasing sampling strategies [YJSL05, LL04], decomposing environments and focusing sampling on critical paths [GFC09], taking into account the optimality of solutions [KF10], etc.

2.2 Handling Narrow Passages for Rigid Body Robot

One of the most difficult challenges for sampling-based methods is to efficiently handle narrow passages in the free space of a robot. Uniform sampling commonly used in both PRM and RRT has been identified to show poor performance on environments with narrow passages or bug trap shapes [LaV06]. Many strategies have been proposed to improve the performance on these difficult problems [HLK06].

At a high level, there have been two groups of PRM or RRT variations. One group focuses on diversifying expansions of local planners such as penalizing nodes for extension by recording expansion success rate [CL01, KE05], reducing already-explored space expansions by a regression-prevention mechanism [KvdP06], and regulating expansions in an adaptive manner [WB11b, WB11a].

Another group tries to alter the sampling distribution to efficiently guide the expansion of roadmaps or trees inside narrow passage regions by the use of the workspace geometric information [KH06, RTPA06], dynamic sampling distributions [MTP$^+$05], simultaneously mapping roadmaps at C-free and C-Obstacle spaces [DA11], utilizing local free space information [DL11], filtering (or adaptive sampling) techniques towards important regions including narrow passages [ABD$^+$98, BOvdS99, SLN00, HJRS03, YJSL05, SHJ$^+$05, YTEA12, ST11], and retraction-based approaches [RTLA06, RL05, ZM08, HSAlCL06, SL05].

2.2.1 Filtering techniques

Filtering techniques aim to generate lots of samples at the robot's free space and filter out some of them that are not located near difficult regions such as narrow passages, leading to adaptive sampling. Boor et al. [BOvdS99] proposed the Gaussian sampling strategy that distributes samples near the boundary of the free space. The Bridge test proposed by Hsu et al. [HJRS03] uses three sampled configurations to boost the sampling density inside narrow passages. Yershova et al. [YJSL05] proposed a dynamic-domain RRT that adapts its sampling domain to bias toward the visibility domain by obstacles. Obstacle-based PRM proposed by Amato et al. [ABD$^+$98] generates a ray from invalid samples to find a surface configuration, and UOBPRM [YTEA12] improves the performance by generating surface configurations uniformly. Shkolnik and Tedrake [ST11] proposed the ball tree algorithm that approximates the reachable free space in a similar way to our non-colliding line-test.

2.2.2 Retraction-based techniques

The main idea of the retraction-based approach is to retract initial samples normally generated from uniform sampling on C-space towards more useful regions such as the boundary of C-Obstacle, the medial axis of the free space, and so on. Its final sample distribution, therefore, is not uniform, but biased toward more useful regions. Some of retraction-based algorithms utilize the contact information for retraction [RTLA06, RL05, ZM08], dilate the free space [HSAlCL06, SL05], or use approximation of medial axes or boundary of the motion for the robot [FL04]. These techniques can efficiently handle narrow passages, however these can run slower than a simple method, when the free space does not have such difficult regions because of their computational overheads for retraction operations.

2.3 Path Planning for Redundant Manipulators

Sampling-based single query motion planning algorithms (e.g., RRT [KL00] and EST [HKLR02b]) have been used to solve various motion planning problems including high-dimensional manipulation planning. For overcoming the high dimensionality of redundant manipulators, many techniques have been proposed to improve the performance of a planner by utilizing the task space, which has much lower dimensions compared to the configuration space of a robot.

Yao et al. [YG07] proposed ATACE, which first generates end-effector paths in the task space, and then tries to track the paths in the configuration space by using an end-effector trajectory tracking planner such as Jacobian-based techniques [MK85, NHY87] as a local planner. Bertram et al. [BKDA06] used a heuristic goal function that estimates a distance from a sample to goals, in order to bias sampling in the configuration space. JT-RRT [VWFS07], proposed by Weghe et al., combined the Jacobian transpose method with the standard RRT exploring the configuration space. Especially this method used a task space goal bias that attempts to connect nodes in the configuration space and the goal end-effector position by a local planner based on the Jacobian transpose method. Diankov et al. [DRF$^+$08b] proposed BiSpace Planning, which explores both configuration and task spaces in a similar manner to the bidirectional RRT approach; two different trees are grown from initial and goal states and the planner periodically attempts to connect them.

While planners mentioned above used workspace goal positions for biasing the configuration exploration so that planners could find a solution effectively, techniques exploiting the task space as its main exploration space have been proposed [ST09, BHG10]. A concept of directly exploring the task space is very helpful especially for a hyper-redundant manipulator, which has many dofs, because the dimension of the task space remains the same, while the dimension of the configuration space increases. As a result, the task space exploration is scalable with respect to the dof of robots. In order to reconstruct samples of the configuration space from ones of the task space, Shkolnit et al. [ST09] uses the Jacobian Pseudoinverse method, which is widely used for the task space control [Lie77]. Behnisch et al. [BHG10] further exploits the concept of the direct task space exploration by utilizing the null space velocity for avoiding obstacles.

Many sampling bias or adaptive sampling techniques have been proposed in order to improve the performance of sampling-based motion planners. The common idea of these techniques is to bias the sampling distribution towards more important regions. Some of them use visibility domain by obstacles [YJSL05], boundary of obstacles in the configuration space [RTLA06, ZM08], regions inside narrow passages [HJRS03, DL11], boundary of obstacles near narrow passages [LKZY12], etc. While a biasing technique in the configuration space has been widely researched, biasing the task space sampling distribution has not been studied well, especially for a redundant manipulator.

2.4 Kinodynamic motion planning problems

The classic approach for kinodynamic motion planning in robotics is to decouple the problem as two stages: firstly solving basic path planning problem by a geometric planner, then computes a global solution near the basic path, satisfying the dynamics and other constraints [BDG85, SD91, LJTM94, LSL99, SB02]. However, these approaches can result in a susceptibility to local minima when intractable paths are generated from the first state, because of limits on forces and torques for the robot in the environments.

In order to cope with the local minima problem, a number of approaches were proposed. Svestka and Overmars [SO97] proposed extension of randomized holonomic planning technique to the non-holonomic planning assuming proper steering method for a system, and LaValle and Kuffner [LK01] proposed generalization of sampling-based method into the state space. Randomized kinodynamic planning have been extended to improve the performance [CFL04, SWT09, PKV10, SK12]. However, the state space dimensions is increased two-fold, the complexity of planning algorithms scales more than holonomic planning. Also, selecting a suitable metric and extension methods still are remained challenging.

Discrete planning methods also have been proposed for simple environments by discretization of the environment or the state space and then applying simple efficient graph search techniques to find a solution path [LFG$^+$05, PK05, KK06, LF09, GSL12]. However, approaches are restricted to simple environments and low degree-of-freedom robots and shows low efficiency in planning for large distances and complex planning.

Chapter 3. Selective Retraction-based RRT Planner for a Rigid Body Robot

3.1 Overview

In this chapter, we propose a novel retraction-based planner for rigid robots, *Selective Retraction-based RRT* (SR-RRT), for a wide variety of environments that have or do not have narrow passages. Our method performs the optimization-based retraction operations in order to explore more important regions. Instead of performing retraction operations exhaustively, we selectively perform retraction operations, only if samples to be retracted are deemed to be around narrow passages. To decide whether a sample is near a narrow passage, we propose a *bridge line-test*, an inexpensive filtering operation, which checks whether narrow passages exist along a line segment. In order to achieve a high accuracy even in high dimensional configuration spaces, we perform principal component analysis (PCA) and generate the line with a higher probability to cross narrow passages. Also, in order to generate more random samples near narrow passages, we present a *non-colliding line-test* that detects wide-open free spaces and cull samples near such spaces (Sec. 3.3).

We have implemented our SR-RRT method integrated with these two line-tests. In order to demonstrate benefits of our method, we have tested SR-RRT with various types of benchmarks that have different characteristics (Sec. 4.4). For these tests we assume free-flying systems because the same assumption is also used for the retraction method. Our method achieves up to 21 times, 31 times and 3.5 times performance improvement (6.7 times, 6.8 times and 2 times on average) over a basic RRT [KL00], a Dynamic-Domain RRT [YJSL05], and an optimization-based retraction method [ZM08], respectively. Moreover, while the basic RRT and the optimization-based retraction method show lower performance than the other tested methods in some benchmarks, our method consistently improves the performance across all the tested benchmarks that have or do not have narrow passages. As a result, we can conclude that our method is more robust and general in free-flying systems than other tested methods.

The contents of this chapter is based on the article presented in ICRA 2012 [LKZY12].

3.2 Backgrounds

In this section we give a brief review on the basic RRT and its variant based on an optimization-based retraction, RRRT [ZM08], which is designed for efficiently handling narrow passages. We then discuss their issues that arise when we apply them to various environments, some portions of which contain narrow passages, but other portions of which do not have such difficult regions.

3.2.1 Review of RRT

A basic RRT algorithm starts with a single or multiple random trees [KL00]. The basic RRT method randomly generates a sample, q_r, in the configuration space and finds its nearest neighbor node, q_n, among nodes of existing random trees. It then attempts to connect q_n with q_r. If q_r is in collision or there are any obstacles in between q_r and q_n, a first in-contact configuration, q_c, that touches the boundary of C-Obstacle is computed along a straight line from q_r to q_n [LaV06, pp. 189–192]. q_c is then added to the tree and is connected with q_n (Fig. 3.4-(a)). The pseudo code of the basic RRT is shown at Algorithm 1.

Algorithm 1 The basic RRT Planner

```
Require: tree $T$
  $T$.AddVertex ($q_{init}$)
  repeat
    $q_r \leftarrow$ RandomState()
    $q_n \leftarrow$ NearestNeighbor($q_r$, $T$)
    $q_c \leftarrow$ RRTExtend($q_n$, $q_r$)
    $T$.AddVertex ($q_c$)
    $T$.AddEdge ($q_n$, $q_c$)
  until a collision-free path between $q_{init}$ and $q_{goal}$ is found
```

It has been known that the basic RRT explores the free space with a bias related to the Voronoi diagram [KL00]. Especially, a probability that a node of a random tree is chosen as the nearest neighbor node is proportional to the volume of the Voronoi region associated with the node.

This basic RRT or its variants, however, can take a prohibitively long planning time, when the free space of a robot contains narrow passages. This is because the volume of regions associated with narrow passages are significantly smaller than other regions. As a result, the probability of exploring and getting out the narrow passage region is quite low.

3.2.2 Review of RRRT

In order to address the problem of the basic RRT, an optimization-based retraction technique, RRRT [ZM08], was proposed recently by Zhang and Manocha. Its pseudo code is shown at Algorithm 2. Its main idea is to iteratively retract a randomly generated in-colliding sample to a more desirable place, which is the nearest boundary of the free space, in other words, contact space. To overcome computational prohibitiveness, RRRT formulates the retraction computation to an optimization problem based on a local contact analysis, while iteratively minimizing an objective function (i.e. a distance metric is used here); for example, Fig. 3.1 shows that $q_{c'}$ is a newly retracted from in-contact configuration from q_c.

OptimizedRetraction $(\cdot)$ used in the pseudo code of Algorithm 2 is performed as follows: Firstly, the closest feature pairs are computed between q_c, the first in-contact configuration from q_n to q_r, and the obstacles. It then computes a new sample $q_{c'}$, which is not in-collision and minimizes the distance to q_r by searching over the local contact space constructed implicitly from the closest feature pairs. These steps are repeated by replacing q_n with $q_{c'}$ until the distance to q_r is converged or the maximum number of iterations has been reached. Even though this optimization technique behaves in a greedy manner, this technique has been demonstrated to work well in environments that have narrow passages. For example, this optimization-based retraction technique shows two times higher performance than the basic RRT in the wiper 1.0 benchmark (Fig. 3.8), which has narrow passages.

3.2.3 Issues with Optimization-based Retraction

The optimization-based retraction technique works quite well with scenes that have narrow passages. This result is achieved by performing extra operations to the basic RRT, such as local contact analysis,

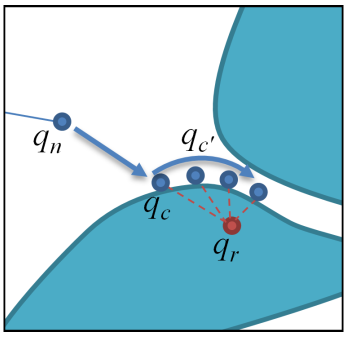


Figure 3.1: Sampling generation of an optimization-based retraction: *RRRT incrementally retracts a randomly generated in-colliding sample q_r to a more desirable place in order to generate more samples in the narrow passages. This figure shows that the original random sample q_r is in-colliding, and it is then incrementally retracted from q_c to $q_{c'}$ and so on.*

Algorithm 2 RRRT Planner

```
Require: tree $T$
  $T$.AddVertex ($q_{init}$)
  repeat
    $q_r \leftarrow$ RandomState()
    $q_n \leftarrow$ NearestNeighbor($q_r$, $T$)
    if HaveCollisionFreePath ($\overline{q_n q_r}$) then
      $q_{new} \leftarrow$ RRTExtend($q_n$, $q_r$)
      $T$.AddVertex ($q_{new}$)
      $T$.AddEdge ($q_n$, $q_{new}$)
    else
      $q_c \leftarrow$ the first in-contact configuration from $q_n$ to $q_r$
      OptimizedRetraction ($q_c$)
    end if
  until a collision-free path between $q_{init}$ and $q_{goal}$ is found
```

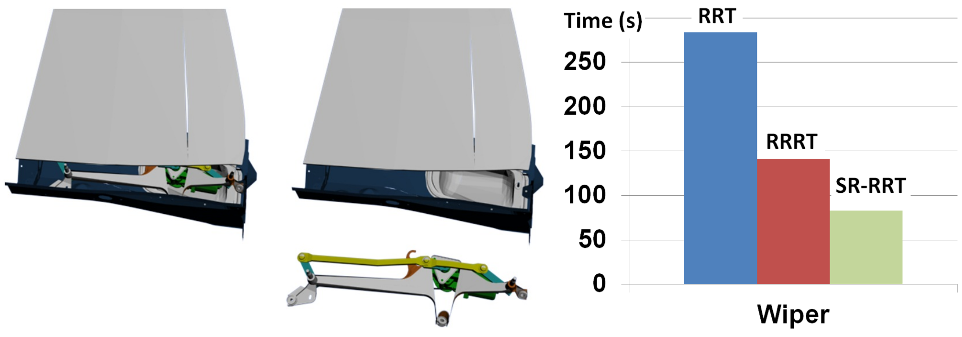


Figure 3.2: Wiper 1.0 Benchmark: *The left two figures show two animation sequences of getting a wiper out of the windscreen model. Since there are not much rooms between two models, this benchmark has narrow passages. The rightmost graph shows the average planning time of different methods performed in 100 times.*

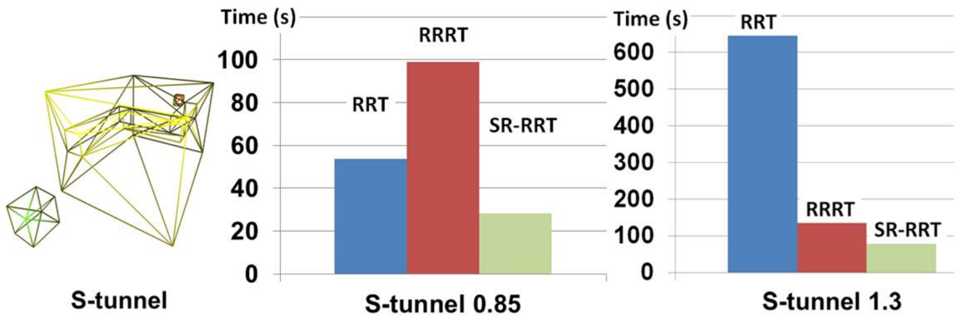


Figure 3.3: S-tunnel Benchmark: *The leftmost figure shows the S-tunnel benchmark. The right two figures show the average performance of two variations of the S-tunnel benchmark. 0.85 and 1.3 indicate the scaling factor of the cubic robot. S-tunnel 0.85 does not include any narrow passages, while S-tunnel 1.3 includes them.*

additional sampling, etc. These operations have relatively higher computational overheads, compared to other operations of the basic RRT method [ZM08]. Also, while this technique explores and gets out of narrow passages efficiently by generating samples on the contact space, it covers all the contact spaces equally well. In other words, this technique tends to generate many samples near the contact space, even though the contact space is not on narrow passages.

As a result, if an environment does not have narrow passages or have many obstacles that create the contact space without generating narrow passages, the optimization-based retraction technique can show even lower performance than the basic RRT, because of both the computational overhead and excessive sampling on the contact space. For example, it shows 84% lower performance than the basic RRT in the S-tunnel benchmark 0.85 (Fig. 3.3) that does not have narrow passages.

3.3 Selective Retraction-based RRT

Our technique is based on the optimization-based retraction method, RRRT [ZM08]. To efficiently handle various types of environments that have or do not have narrow passages, it is critical to identify regions that are likely to be narrow passages and to selectively perform the expensive retraction operation only on those regions.

In order to efficiently identify such narrow passages within our RRT-based planner, we present a *bridge line-test*, which is inspired by the bridge test [HJRS03], originally proposed for probabilistic roadmap techniques. Our bridge line-test uses a line passing through an in-contact configuration to test whether it is likely to have narrow passage around the configuration. We also propose a *non-colliding line-test*, a dual operator to the bridge line-test, to generate more samples near narrow passages.

3.3.1 Selective Retraction-based Extension

We first explain our extension method of SR-RRT, whose pseudo code is at Algorithm 3. Once a random sample q_r and its nearest neighbor node q_n are computed as mentioned in Sec. 3.2.1, then we perform our extension method, **SRExtend** (q_n, q_r), shown in Algorithm 3. The first step of our extension method is exactly the same to the extension method of the basic RRT explained in Sec. 3.2.1. Note that at this step, we may create an in-contact configuration q_c.

Algorithm 3 SRExtend(q_n, q_r)

```
if HaveCollisionFreePath ($\overline{q_n q_r}$) then
  return RRTExtend($q_n$, $q_r$)
end if
$q_c \leftarrow$ the first in-contact configuration from $q_n$ to $q_r$
if BridgeLineTest ($q_c$) then
  OptimizedRetraction ($q_c$)
end if
return $q_c$
```

As the second step of our extension method, we perform our bridge line-test to decide whether we need to perform the retraction operation. If the bridge line-test indicates that there is a narrow passage around the in-contact configuration q_c, we perform the retraction operation and generate another in-contact configuration $q_{c'}$ in a way that we can reduce the distance between q_r and the newly generated in-contact configuration $q_{c'}$. This operation is represented as *OptimizedRetraction (·)* in the pseudo code of our extension method (Algorithm 3) and explained in Sec. 3.2.2.

3.3.2 Bridge line-test

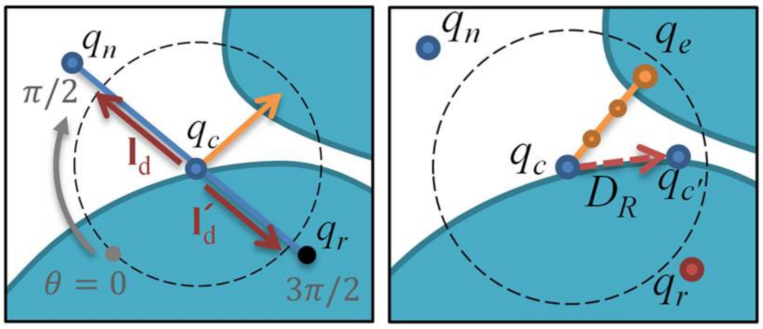
The bridge line-test, **BridgeLineTest** (q_c), determines whether a narrow passage exists around an in-contact configuration q_c. To perform the bridge line-test, we first generate a line whose one end is located at q_c. The line can have an arbitrary line direction. Instead of generating the line direction in a uniform manner, we take into account available local information around q_c to make the line direction to cross narrow passages with higher probability.

Note that it has been identified that a collision-free path exists from q_c to q_n (see the left image of Fig. 3.4-(a)). As a result, we can assume that there are no narrow passages along that particular direction, $\vec{l_d}$, which is $q_n - q_c$. On the contrary, a line segment between q_c and q_r is in C-Obstacle. As a result, we can also assume that there are no narrow passage along a line direction, $\vec{l'_d} = (q_r - q_c)$, from q_c. This local information leads us to generate an arbitrary line direction with a higher probability in these intervals defined between these two line directions, to make the randomly generated line to cross potentially-existing narrow passages.

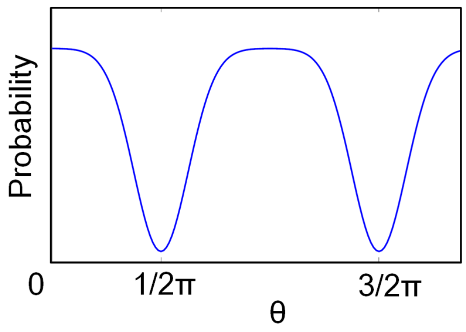
Generating a bridge-line

In particular, we use a line generating function as a probability distribution function (PDF), $p_l(\cdot)$, a mixture of two reflected Gaussians, each of which has near zero probability at these two line directions $\vec{l_d}$ and $\vec{l'_d}$, while they peak at a plane whose normal is either $\vec{l_d}$ or $\vec{l'_d}$; Fig. 3.4-(b) shows the 2D example of this distribution function.

Once we generate a line direction starting from q_c, then we compute the other end point, q_e of the line by computing a random distance, d, according to another probability function, $p_d(\cdot)$. We use the Gaussian function whose mean is the average distance between successive in-contact configurations computed by the retraction operation (e.g., D_R in Fig. 3.4-(a)). We set the standard deviation of the Gaussian function to be small, but to have a meaningful probability (e.g., $D_R/2$) to identify very small narrow passages; we make sure that $p_d(0)$ has a non-zero probability to detect narrow passages with infinitely small width,



(a) Procedure of the bridge line-test



(b) PDF for the line direction

Figure 3.4: *(a) Two figures show the procedure of generating and performing a bridge line-test. (b) A probability distribution function (PDF) for the line direction parameterized with θ.*

After computing a line whose two end points are q_c and q_e, i.e. $\overline{q_c q_e}$, we check whether there is a narrow passage in the line. Specifically, the narrow passage is defined by collision regions at both ends of the line with a free space in the middle of the line. Since q_c is an in-contact configuration, we mainly check whether there are a free space in the middle and collisions in the other end q_e of the line. If we have such a case, in other words, there is a bridge line that connects two obstacle regions, we treat the region around these two configurations of q_c and q_e as they are in a narrow passage, and thus perform the retraction operation. We ignore the case where the tested line is located in obstacles, i.e. penetrates the obstacles, since this is not considered as to be within a narrow passage. For detecting collisions on the line, we can use either continuous or discrete collision detection methods. For discrete collision detection methods, we check a fixed number of intermediate configurations on the line.

Re-testing

Our bridge line-test can fail with a low probability to identify a region to be in a narrow passage, even though the region is in a narrow passage. This failure is mainly because our bridge line-test considers only one dimensional line in multiple dimensional configuration spaces to check whether a node is in a narrow passage. Instead of performing the bridge line-test only one time for each node, we perform the bridge line-test, whenever an in-contact configuration that was not identified as narrow passages with earlier bridge line-tests is chosen as a nearest neighbor node of a random sample. This re-testing is defined as **Re-test**(q_n) shown in Algorithm 4.

Note that it is quite reasonable to re-test the node with the bridge line-test, since the node selected as a nearest node to others many times implies that the tree expansion may be stuck there by potentially-

Algorithm 4 Re-test(q_n)

```
if $q_n$ is in-contact configuration and did not pass earlier bridge line-tests then
  if BridgeLineTest ($q_n$) then
    OptimizedRetraction ($q_n$)
  end if
end if
```

existing narrow passages around the node. Moreover, by allowing multiple re-testing, the bridge line-test can correctly identify the existence of narrow passages in a probabilistic manner with many random samples.

3.3.3 High-Dimensional Configuration Spaces

Our bridge line-test is very efficient, since the line-test considers whether there are collisions between the line of robot configurations and the environment. However, its accuracy degrades as the dimension of the configuration spaces goes higher because of its one dimensional nature of checking collisions. In order to ameliorate this problem, we consider how local free spaces grow, and generate random lines of bridge line-tests more frequently in dimensions that may contain narrow passages.

Inspired by a recent dimension reduction technique developed for random motion planning approaches [DL11], we perform the principal component analysis (PCA) [Jol86] to see how local free space is distributed, and treat a region to be in a narrow passage when the local free space is not uniformly distributed in the configuration space. For example, if the free space is located in a narrow passage, we can assume that the free space is not uniformly distributed, but is severely constrained and thus has an elongated shape (see the left image of Fig. 3.5).

The PCA is a well-known statistical procedure that transforms a set of points to a new coordinate system whose first axis corresponds to the direction with maximum variance of the input data and second axis maximizes the variance in subspace orthogonal to first axis and so on [Jol86]. The PCA is commonly used for dimensionality reduction preserving most of the information. The PCA can be computed by using the covariance matrix of input points and finding the eigenvectors and their corresponding eigenvalues of the matrix. The computed eigenvectors are treated as principal components of input points.

We apply the PCA in a similar manner as used in the PCA-RRT [DL11]. When we need to generate a random line from q_c, we first collect k nearest neighbors from q_c by a breadth-first search, and then perform the PCA on those nearest neighbors. We assume that the principal component with the maximum eigenvalue, i.e. variance, aligns with the longest axis of the elongated-shaped narrow passage. Our goal of choosing the random line direction is to increase a probability that the generated direction crosses the longest axis of the narrow passage.

To meet our goal, we transform $\vec{d_r}$, computed in the line generating PDF $p_l(\cdot)$ in Sec. 3.3.2, into a new line direction $\vec{d'_r}$ based on the following equation:

$$\vec{d'_r} = \sum_{i=1}^{k} (\frac{1}{\lambda_i} \vec{d_r} \cdot U_i) U_i,$$

where U_i is i-th eigenvector and λ_i is its corresponding eigenvalue. This equation transforms $\vec{d_r}$ to follow principal components more that have lower variances, leading to crossing a potentially-existing narrow passage, since $1/\lambda_i$, a transforming weight, becomes bigger as we have lower variances.

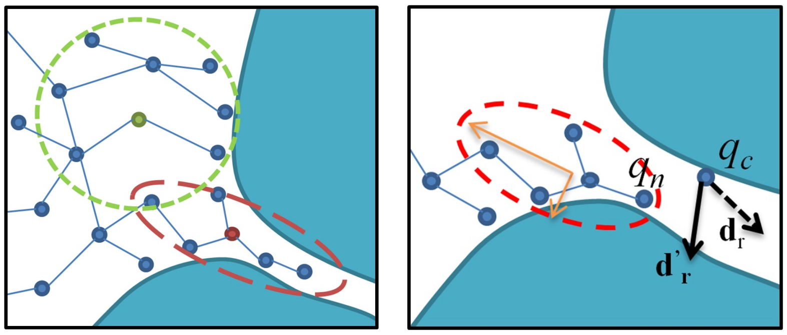


Figure 3.5: *The left figure shows shapes of local free spaces (green and red ones) computed from two nodes. The red one located in a narrow passage has elongated shape, while the green one located in wide-open free space is uniformly distributed in the space. The right figure shows the transformed line direction $\vec{d'_r}$ from an initially generated direction $\vec{d_r}$. Note that orange vectors are eigenvectors scaled by the corresponding eigenvalues computed from the local region of q_n.*

The accuracy of the PCA-based technique for transforming the line direction depends on how well our assumption is valid given a local configuration. Moreover, this technique makes a bias for generating line directions for our bridge line-test. In order to mitigate the negative effects of our PCA-based technique, we adopt the transformed line direction $\vec{d'_r}$ with a probability $p_l(\vec{d'_r})$, the line generating PDF. If $\vec{d'_r}$ is rejected, we generate a line direction according to $p_l(\cdot)$ as mentioned in Sec. 3.3.2. As a result, the PCA-based bias has $(1-p_l(\vec{d'_r}))p_l(\vec{d'_r})$ higher probability over our prior line generating function $p_l(\cdot)$, given the PCA-based computed line direction $\vec{d'_r}$.

3.3.4 Non-Colliding line-test

We can generate more samples near narrow passages by using both the bridge line-test and the optimization-based retraction operator. However, these retraction operations can be performed, once a random sample is located closely to such regions. In order to increase the probability to generate samples near such regions in an efficient manner, we propose a sampling bias technique using the *non-colliding line-test*, which is a dual operator to our bridge line-test.

Our main idea is that once we identify wide-open free spaces in the configuration space, it is more desirable to discard samples generated within such free spaces and to generate more samples outside those areas and potentially towards narrow passages.

It is, unfortunately, challenging to exactly define wide-open free spaces in high-dimensional configuration spaces. Instead of constructing them deterministically, we define them in a probabilistic manner. At a high level, we generate a line segment from a node (similar probabilistic manner used in the bridge line-test) and check whether the line is a collision-free path. If so, we treat the region around the node as a wide-open free space.

Let us first define a *free hypersphere* in the configuration space to be a hypersphere, whose contained configurations do not have any collisions against the environment. Specifically, we define the center of each hypersphere to be located at a node of the random tree, except for in-contact configurations; in-contact configurations have contacts by the definition and thus we do not consider them as wide-open free spaces. Also, we set the radius of each hypersphere to be the distance computed between the center node of the hypersphere and its nearest neighbor node (Fig. 3.6). Note that when we add a new node to the random tree, we compute the distance, d_{NN}, between the new node and its nearest neighbor node,

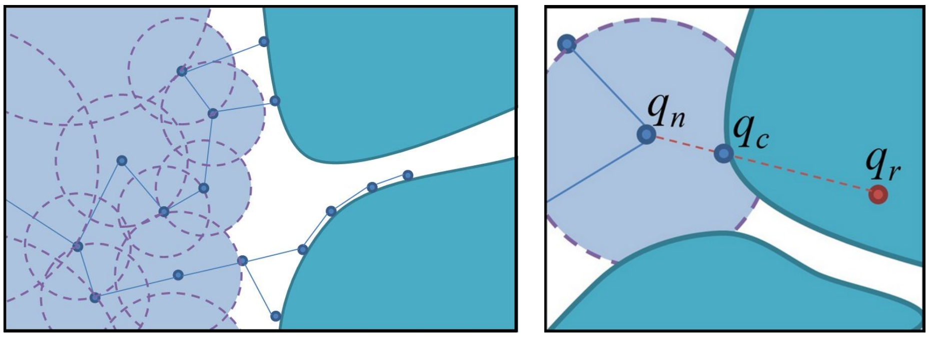


Figure 3.6: *The left figure shows free hyperspheres that approximate wide-open free spaces. Dotted circles represent to the hypersphere with the center of node and radius of it. The right figure shows an example that a node q_c is generated from a random sample q_r, generated outside of hypersphere (dotted circle) by a basic RRT expansion.*

and use the distance for the radius of each hypersphere associated with the new node and its nearest neighbor node.

In order to determine whether a hypersphere of a node is free hypersphere or not, we use the non-colliding line test in order to approximate. **NonCollidingLineTest**(q_n) generates a random line starting from the center node, q_n, of the hypersphere. If there are no collisions in the random line, we treat the hypersphere to be free hypersphere. Note that the non-colliding line- test acts as an efficient probability function in terms of identifying whether a region can be classified as a wide-open free space; even though performing the non- colliding line-test one time may incorrectly identify a hypersphere as a free hypersphere, performing it whenever a node is chosen as the nearest neighbor node can identify a region correctly in a probabilistic manner.

To generate a line used for a non-colliding test, we uniformly generate a random direction for the line segment starting from the center node q_n of a hypersphere. Then we compute another end point, q_e, of the line along the chosen line direction. q_e is computed by a random distance generated by a Gaussian distribution function, whose mean and standard deviations are set to be the half of the radius of the hypersphere associated with q_n.

Once we have a set of approximated free hyperspheres, we discard a random sample if the random sample is within the free hypersphere of its nearest neighbor node.

3.3.5 Overall algorithm

A pseudo code of the overall algorithm of our SR-RRT planner is shown at Algorithm 5. We randomly generate a sample q_r and find its nearest neighbor node q_n. Then we discard it if the random sample q_r is inside the wide-open free spaces probabilistically defined by performing the non-colliding list-test. Otherwise, we perform our extension algorithm (Algorithm 3) after performing the re-testing process (Algorithm 4). As the last step of our method, we update the nearest neighbor distance, d_{NN}, and the tree, T. We iteratively perform these steps until we find a collision-free path between the initial and goal configurations.

Algorithm 5 SR-RRT Planner

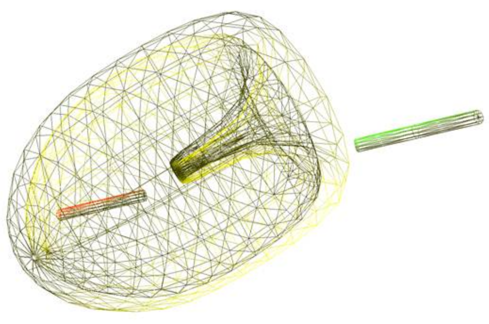
```
Require: tree $T$
  $T$.AddVertex ($q_{init}$)
  repeat
    $q_r \leftarrow$ RandomState(); $q_n \leftarrow$ NearestNeighbor($q_r$, $T$)
    if $q_n$ is not an in-contact AND dist($q_n$, $q_r$) $< q_n.d_{NN}$ then
      if NonCollidingLineTest($q_n$) then
        CONTINUE
      end if
    end if
    Re-test($q_n$)
    $q_{new} \leftarrow$ SRExtend($q_n$, $q_r$)
    if $q_n.d_{NN} >$ distance($q_n$, $q_{new}$) then
      $q_n.d_{NN} \leftarrow$ distance($q_n$, $q_{new}$)
    end if
    $q_{new}.d_{NN} \leftarrow$ distance($q_n$, $q_{new}$)
    $T$.AddVertex ($q_{new}$); $T$.AddEdge ($q_n$, $q_{new}$)
  until a collision-free path between $q_{init}$ and $q_{goal}$ is found
```

3.4 Experimental Results

We have implemented SR-RRT for three dimensional rigid body robots on an Intel i7 desktop machine that has 3.33GHz CPU. Our method is built upon a basic RRT integrated with an efficient optimized-based retraction method, RRRT [ZM08]. For the local planning, we use a linearly interpolated motion between two configurations and check whether we have collisions on a fixed number of intermediate configurations on the linearly interpolated motion.

3.4.1 Benchmarks

We test our method against ten benchmarks that have different characteristics. We classify our benchmarks as three types in terms of relative difficulty: environments with a tight narrow passage and thus requiring longer computation time to find a solution (Hard), environments consisting of relatively wide spaces, i.e. environments that are relatively easy to solve (Easy), and in-between environments (Moderate). The S-tunnel model (Fig. 3.3) has a S-shape of tunnel and we change its characteristics by scaling a cubic-shaped robot; S-tunnel x uses a scaling factor of x for the robot. Benchmarks with an easy type include S-tunnel 0.85, whose robot is quite small enough to pass through the tunnel easily. Benchmarks with a hard type include S-tunnel 1.3, bug-trap (Fig. 3.7-(a)), flange (Fig. 3.7-(b)), wiper 1.0 (Fig. 3.8), pipe (Fig. 3.9), and living room (Fig. 3.10), which all include narrow passages in environments. Finally, S-tunnel 1.0 and wiper 0.9 benchmarks belong to the class of moderate; Wiper x uses a scaling factor of x for the robot (wiper). The dimensions of the configuration spaces in all of these benchmarks are six. The benchmark information is summarized in Table 3.1.


(a) Bug-trap

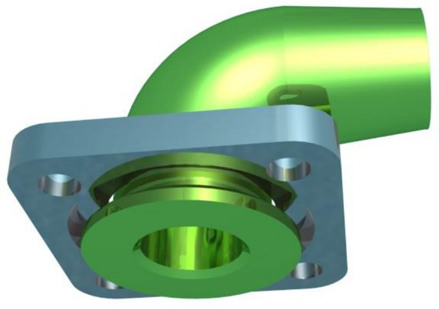

(b) Flange

Figure 3.7: *This figure shows two different benchmarks that have narrow passages. Note that a robot of a bug-trap benchmark is initially located outside of a trap.*

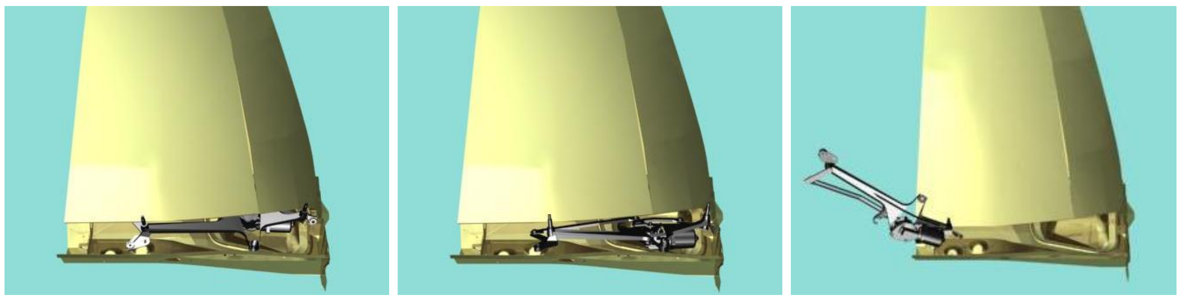


Figure 3.8: *Three image shots that show animation sequences of getting a wiper out of the windscreen model. Since there are not much rooms between two models, this benchmark has narrow passages.*

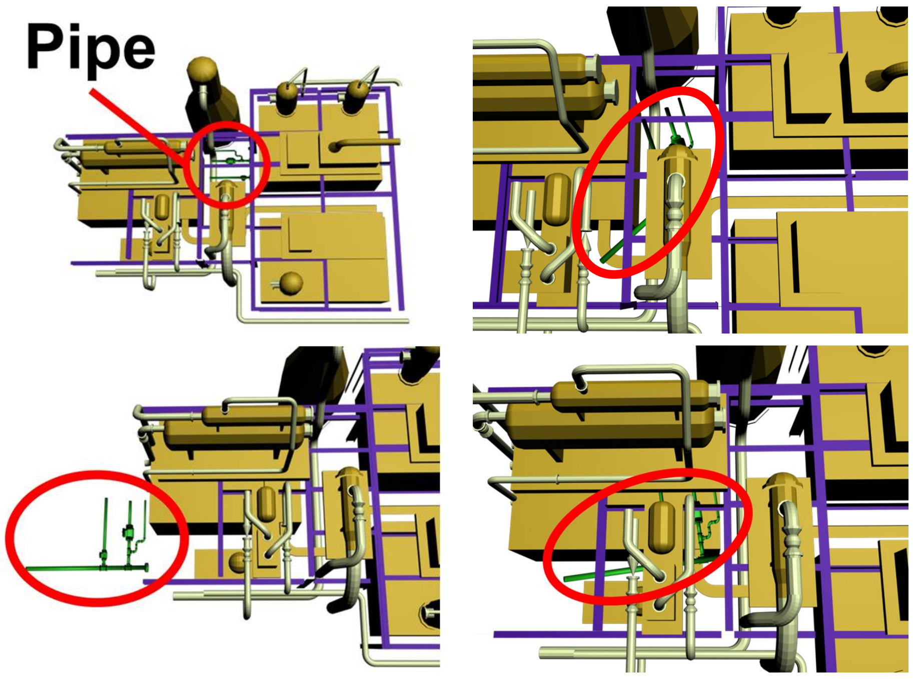


Figure 3.9: *Four image shots are shown for moving the pipe out of an industrial environment (clockwise starting from the top left).*

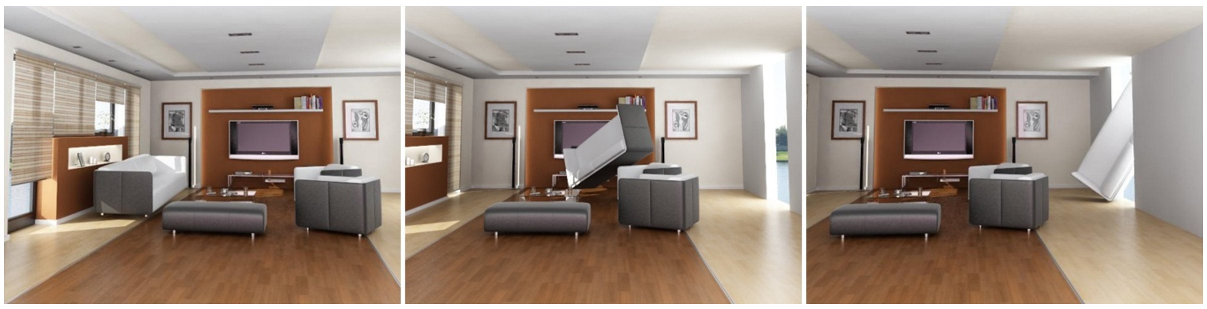


Figure 3.10: Living room benchmark: *This figure shows three image shots of getting a sofa out through a door, which creates narrow passages in 6-dofs configuration space. Our method achieves ten and two times improvements over a basic RRT and an optimization-based retraction method, respectively.*

3.4.2 Results and comparisons

In order to demonstrate the relative benefits of our method, we have also implemented the basic RRT, called RRT, and the optimization-based retraction RRT, called RRRT, which are described in Sec. 3.2.1. We use the same values for parameters (e.g., the collision checking frequency used in a local planner) that are shared between different versions of RRT methods. We run all the methods including ours, SR-RRT, with each of our benchmarks in 100 times and report the average running time in Table 3.1 for a fair comparison by removing the randomness in the performance inherited from the random sampling procedure.

For all the benchmarks, our method computes collision-free paths in less than three minutes. In the pipe and living room benchmarks, our method spends over one minute on average to compute a collision-free path, since these models consist of more than 40 K triangles and have narrow passages.

For the Hard-typed benchmarks, our method shows higher (e.g., 72% higher on average) performance over RRRT and much higher (e.g., 7.7 times higher on average) over RRT. Since RRT does not bias its sampling towards narrow passages, it runs quite slowly in Hard-typed benchmarks. RRRT shows higher performance than RRT. However, because of its higher computational overheads caused by performing retractions on all the contact spaces, RRRT runs slower than our method.

For the Easy-typed benchmarks, RRRT shows lower (e.g., 46% on average) performance over RRT, because RRRT excessively generates many in-contact configurations, most of which do not capture a new connectivity of free spaces, but pose the computational overheads. On the other hand, our method still shows 91% improvements on average over RRT. Even though its improvement over RRT is weaker than improvements made with Easy-typed benchmarks, our method shows improvements over RRT even in these benchmarks that do not have narrow passages. Furthermore, our method shows 3.51 times improvements over RRRT, since our method selectively performs retraction operations. These results indicate that the overheads of our line-tests are small and demonstrate the robustness of our methods.

For the Moderate-typed benchmarks, RRRT shows slightly lower (e.g., 15% on average) performance over RRT, while our method still shows 2.20 times improvement over them. Therefore, we can conclude that our method works more robustly than RRT and RRRT for a wide variety of environments that have or do not have narrow passages.

We have tested the Dynamic-Domain RRT [YJSL05] (DD-RRT) with all the benchmarks which adapts is sampling domain to bias toward the visibility domain by obstacles. DD-RRT shows overall 1.57 times performance improvement over the basic RRT. Although performance of DD-RRT is very

Table 3.1: *Performance of our method, SR-RRT (***SR-RRT***), the basic RRT (***RRT***), the Dynamic-Domain RRT [YJSL05] (***DD-RRT***) and the optimization-based retraction RRT (***RRRT***) [ZM08] for each benchmark model with its representative image, and model complexity, and difficulty level. The table also shows improvements (Imp.) of our method over* **RRT**, **DD-RRT** *and* **RRRT**. *See Sec. 4.4 for the detailed information.*

Model	# Tri	Difficulty	RRT	DD-RRT	RRRT	SR-RRT	Rep. Image
S-tunnel 0.85	64	Easy	53.68	58.78	98.83	28.14	Fig. 3.3
S-tunnel 1.0	64	Moderate	96.21	129.92	141.22	65.57	See above
S-tunnel 1.15	64	Hard	406	405.93	129.5	52.08	See above
S-tunnel 1.3	64	Hard	646	849.92	134.72	78.9	See above
Bug-trap	2.7k	Hard	145	35.45	39.72	22.17	Fig. 3.7-(a)
Flange	6.3k	Hard	589.24	875.25	33.12	27.99	Fig. 3.7-(b)
Wiper 0.9	26.7k	Moderate	107.57	63.99	107.75	48.05	Fig. 3.8
Wiper 1.0	26.7k	Hard	283.56	143.27	141.5	82.91	See above
Pipe	48.4k	Hard	639.99	1001.97	225.79	166.08	Fig. 3.9
Livingroom	137k	Hard	776.54	242.94	140.94	77.2	Fig. 3.10

dependent on environments, our planner shows better performance with all the benchmarks with or without narrow passages. Our planner shows 6.8 times performance improvements overall compared to DD-RRT (See Table 3.1).

Table 3.2 shows the number of iterations of RRT, RRRT, and SR-RRT needed to compute a collision-free path for benchmarks. For all the benchmarks, RRT takes the largest number of iterations, while RRRT takes the smallest number of iterations, and SR-RRT takes the middle. Overall one iteration of RRRT (0.0040 sec) shows 4 times slower performance than one iteration of RRT (0.0168 sec). Our planner, SR-RRT, reduces the number of retraction iterations, which takes relatively much time, and increases the number of a basic RRT-like iterations exploring C-space. Overall, the total number of iterations of ours is less than the one of RRT, and the average running time of one iteration (0.0042 sec) is similar with RRRT. As a result, our method shows higher performance than RRT and RRRT.

3.5 Discussions

In this section we discuss a few important issues of our method.

3.5.1 Contributions of Each Component

We measure how much improvement we make with each component of our contributions. By enabling bridge line-tests (Sec. 3.3.2) only, we observe 74.6% improvement over RRRT. We achieve 5.48% further improvement on average by additionally enabling non-colliding tests (Sec. 3.3.4). Also, by adopting the PCA-transformed line direction (Sec. 3.3.3), we observe additional 5.04% improvement on average. Table 3.2 shows incremental effects by enabling each component on each tested benchmark.

The effectiveness of our PCA based operations is smaller than that of the bridge line-test in our

Table 3.2: *the number of iterations and The contribution of each component on SR-RRT (incremental effects by enabling each component on each tested benchmark. BL-test, NC-test and PCA represent to the Bridge-line test(Sec. 3.3.2) , Non-colliding line-test(Sec. 3.3.4), and PCA-transformed line direction selection(Sec. 3.3.3), respectively.*

Model	**RRT**	**RRRT**	**SR-RRT**	**BL-test**	**+ NC-test**	**+ PCA**
S-tunnel 0.85	17772	5034	8418	35.45	28.83	28.14
S-tunnel 1.0	40692	6259	21545	78.56	70.01	65.57
S-tunnel 1.15	112507	10914	24994	53.66	53.26	52.08
S-tunnel 1.3	233123	21425	56320	81.08	80.89	78.9
Bug-trap	51748	4512	8715	30.47	26.4	22.17
Flange	31571	715	1074	32.32	30.06	27.99
Wiper 0.9	281571	65124	110843	50.87	53.51	48.05
Wiper 1.0	683262	12047	312008	90.64	86.15	82.91
Pipe	378588	8142	144896	176.42	170.6	166.08
Livingroom	152451	12741	55920	94.15	84.17	77.2
Average	198328	14691	74473	72.36	68.39	64.91

tested benchmarks. This is mainly because we have tested only free-flying robots that have six dofs. The PCA operation would be more effective for high-dimensional robots, where accuracy of the bridge line-test would decrease. The non-colliding line-test is effective, when the configuration space consists of widely open free spaces. For example, we observe about 20% improvement in S-tunnel 0.85, where a robot is small and most of space are relatively wide open free spaces. Our tested benchmarks are mostly hard-typed ones that include narrow passages and less widely open free spaces.

3.5.2 Breakdown of running time

Table 3.3 shows a breakdown of the running time of our method for different components including the retraction, two types of line-tests, and other parts (e.g., computing in-contact configurations, connecting nodes, etc. related to the basic RRT); time spent for performing PCA is included in the bridge line-test. Depending on benchmarks components take different portions of the overall running time. In most of the benchmarks, two line-tests take about 7% to 17% of the overall running time. For the bug-trap benchmark, these two tests take 37% of the overall running time, since many in-contact samples are chosen as nearest neighbors for the tree expansion. Still retractions and basic RRT operations take much larger portions than our two tests.

The culling ratios of samples due to the non-colliding line-tests are quite high (e.g., 78% to 97%) across all the benchmarks. On the contrary, the culling ratios of retraction operations due to bridge line-tests relatively vary a lot. The flange benchmark shows the lowest culling ratio (e.g., 35%) because there are a lot of narrow passages, while getting the flange out of the curved pipe. In the S-tunnel 1.0 benchmark, we achieve up to 98% culling ratio, since the benchmark does not have any narrow passages.

Table 3.3: *Breakdown of the running time of SR-RRT. Cull % refers to the culling ratios of two types of line-tests.*

Model (scale)	**Flange**	**S-tunnel** 1.0	**S-tunnel** 1.3	**Bugtrap**	**Wiper** 1.0	**Pipe**	**Room**
Retraction	76%	24%	52%	51%	57%	50%	31%
BLTest	6%	13%	14%	28%	12%	6%	14%
NCTest	1%	3%	3%	9%	2%	1%	6%
RRT	17%	60%	31%	12%	29%	43%	49%
Cull % of BLTest	35%	98%	83%	51%	72%	74%	83%
Cull % of NCTest	87%	86%	84%	78%	97%	90%	89%

3.5.3 Statistical error of two line-tests

Proposed two line-tests, bridge and non-colliding line-tests, are approximations of narrow passage and free hypersphere detection, respectively. Although we have shown that these tests work successfully with our tested benchmarks in a probabilistic manner, both tests have certain statistical errors.

In the bridge line-test, the probability of making a type I error, a false positive, is zero, because a bridge line is never found when there is no narrow passage (Sec. 3.3.2). On the other hand, we can have a type II error, a false negative. We designed our method to have these characteristics, since a type I error causes unnecessary retraction operations and lowers the overall performance, but an additional computational overhead for a type II error is relatively small.

On the other hand, the probability of making a type II error, a false negative, is zero and a type I error can occur in the non-colliding line-test (Sec. 3.3.4). When a type I error occurs, it rejects a sample inside of a non-free hypersphere that may contain important regions and be useful for expanding a random tree (e.g. an entrance of a narrow passage). The influence of this error is, however, reduced as we generate nodes within hyperspheres.

Specifically, the influence of this error is limited by its corresponding hypersphere, whose radius is set to be the distance from the nearest neighbor in the tree to the center of the hypersphere. As a result, each hypersphere associated with a node is getting smaller and thus its influence of error is also reduced. Although samples inside of hyperspheres can be rejected by an error of the non-colliding line-test, nodes can be created inside of hyperspheres from samples generated outside of any hyperspheres by constructing in-contact configurations during a basic RRT expansion (Sec. 3.2.1). The right figure of Fig. 3.6 shows an example.

3.5.4 Implementation of PCA operations

Table 3.4 shows portions of time spent for performing PCA compared to the overall running time and the bridge line-test. We implemented an incremental PCA [ERP$^+$04], in order to reduce the running time of PCA operation by eliminating re-computation of the diagonalization of the covariance matrix [DL11]. The PCA operation still takes a large portion of the bridge line-test, but its portion of the overall running

Table 3.4: *Breakdown of the running time of PCA operations.*

Model (scale)	**Flange**	**S-tunnel** 1.0	**S-tunnel** 1.3	**Bugtrap**	**Wiper** 1.0	**Pipe**	**Room**
% of overall	1.78	6.59	7.29	14.17	6.27	2.85	7.66
% of BLTest	29.6	50.7	52.1	50.6	52.2	47.6	54.8

time is relatively low (6% on average).

A parameter k, the number of nearby nodes for performing PCA, should be large enough to approximate a local shape of the free space. We have tested different parameter values and found that 10 to 30 for k works reasonably well and shows similar performance.

3.5.5 Implementation issues

When we compute a probability distribution function $p_d(\cdot)$ to sample an end-point of a bridge line-test (Sec. 3.3.2), we use the Gaussian function with the mean used for an optimization-based retraction operation, D_R. The main reason of choosing the mean in such a way is that since we go deeper on average in the amount of D_R in a narrow passage with a retraction operation, we aim to identify narrow passages with that amount of width. A uniform distribution function also could be used for $p_d(\cdot)$, but we have observed that the accuracy of detecting a narrow passage is decreased.

In order to perform our two types of line-tests, we use a discrete collision detection method, which checks for collisions in a fixed number of intermediate configurations on a line. For the frequency of checking collisions for our line-tests, we simply use the same resolution of employed in the local planner.

3.5.6 Analysis with Varying Scaling Factors

We have tested S-tunnel models with varying scaling factors for the cubic-shaped robot (Table 3.1). As we increase the scaling factor, the benchmark poses a more challenging narrow passage problem. In this setting, our method achieves noticeably higher performance improvements over RRT, as we have more challenging narrow passage problems (e.g., 1.9, 7.8, and 8.19 times improvements over S-tunnel 0.85, 1.15, and 1.3 respectively). On the other hand, our method shows slowly diminishing improvements over RRRT as we increase the scaling factor. This is mainly because there are more narrow passages and thus culling ratio of our line-tests decreases. For example, the flange benchmark has narrow passages in most of its free space. As a result, our method shows almost similar, but still higher performance to that of RRRT. These results also demonstrate both the low computational overheads and robustness of our method.

3.5.7 Limitations

Our method works quite well with all the tested benchmarks. Even though the proposed line-tests with PCA computations can be performed without much overheads, their accuracy in terms of identifying narrow passages and wide-open areas may not be high in other scenes. This is mainly because we check a fixed number of configurations on a line in the configuration space. Also, even though there are no narrow passages, our bridge line-tests may treat sharp corners as narrow passages. Our method does not guarantee to always improve the performance over the basic RRT and the optimization-based retraction RRT, because of these properties. Also, even though we identify narrow passages, they may not contribute to the final solution. Nonetheless, among all the tested benchmarks, our method shows improvements over other tested RRT methods, because of its low computational overheads and probabilistically high accuracy.

Chapter 4. Productive Regions Oriented Task space path planning for hyper-redundant manipulators

4.1 Overview

In this chapter, we propose a novel sampling strategy to improve the performance of a trajectory planner that directly explores the task space for hyper-redundant manipulators. For our sampling method, we propose to use a concept of productive regions, which effectively guide a random tree towards a goal state. We then propose two sampling domains for each node in the random tree: a promising maximum reachable area (MRA) and a detour sampling domain. We construct an MRA of a node in the task space such that it approximates a region that a manipulator can reach from the node by using an employed local planner (e.g., a feedback controller with inverse kinematics solutions [ST09]) without any collisions. When the MRA of a node contains the goal state, we classify it as a promising node and use its MRA as its sampling domain. For other nodes that do not contain the goal state, we call them unpromising and additionally construct a detour sampling domain for detouring obstacles constraining the manipulator.

To show benefits of our method, we compared our method against the standard RRT and the Task-Space RRT (TS-RRT) [ST09] across various dofs manipulators, which have $\mathbb{R}^2$ task space. Our experimental results show that our planner gives a solution on complex scenes with many obstacles faster by a factor of up to 3.54 than TS-RRT, which is the state-of-the-art task space planner. This improvement is mainly caused by effectiveness and efficiency of our sampling bias technique that covers productive regions towards the goal state. In addition, we have applied our method to a motion planning problem of cabled mobile robots by approximating a cabled robot to a manipulator with high dofs. Experimental results show that our planner gives higher success ratio (2.3 times) and better performance (3.6 times) for a complicated environment than TS-RRT in this application.

4.2 Planning Algorithm

In this section, we give a brief overview of our global planner and its local planner as a preliminary. We then discuss issues that arise when a task space distance function is used for the planner.

4.2.1 Global Planner

Our global planner is based on the Task-Space RRT (TS-RRT) [ST09]. TS-RRT is similar to the standard RRT, but performs its sampling procedure in the task space. TS-RRT iteratively grows a tree whose nodes consist of both task space and configuration space states. In each iteration, TS-RRT generates a random sample, x_r, in the task space and finds its nearest neighbor node from its random tree. The nearest neighbor node is associated with $[x_n, \, q_n]$, which encodes its task space and configuration space states, respectively. An action u is then computed by a local planner (See 4.2.2) to connect the nearest neighbor node and the random sample in the task space. Finally, a new configuration space state, q_{new} is computed by taking the action u from q_n. If there is no collision in between q_{new} and q_n, then a new node $[x_{new}, q_{new}]$ is added to the random tree, solving the forward kinematics equation with q_{new} to get the corresponding task space state x_{new}. Its pseudocode is shown in Algorithm 6.

Algorithm 6 Task-Space RRT, TS-RRT

```
Require: tree $T$
  $T$.AddVertex ($x_{init}$, $q_{init}$)
  repeat
    $x_r \leftarrow$ RandomStateInTaskSpace()
    $[x_n, q_n] \leftarrow$ NearestNeighbor($x_r$, $T$)
    $u \leftarrow$ Control($[x_n, q_n]$, $x_r$)
    $q_{new} \leftarrow$ NewState($q_n$, $u$)
    if CollisionFree($q_{new}$, $q_n$) then
      $x_{new} \leftarrow$ ForwardKinematics($q_{new}$)
      $T$.AddVertex($[x_{new}, q_{new}]$)
      $T$.AddEdge($[x_n, q_n]$, $[x_{new}, q_{new}]$)
    end if
  until a collision-free path between $x_{init}$ and $x_{goal}$ is found
```

4.2.2 Local Planner

In sampling-based algorithms, a local planner is responsible for connecting two nodes in a sampling space. Since sampling is performed on the low dimensional task space in the global planner, the local planner should generate full trajectories in the configuration space following their task space trajectories. A naive way for generating such trajectories in the configuration space is to generate a random action until a manipulator reaches to its local goal. Unfortunately, this is very inefficient, especially for hyper-redundant manipulators whose configuration space has quite high dimensions. A better and well established approach is to exploit a feedback controller with inverse kinematics solutions [ST09].

Among many models to solve the inverse kinematics problem, the Jacobian pseudoinverse method is widely used in the control literature [SK08, pp. 245–268]. The Jacobian pseudoinverse solution for a configuration space velocity vector $\dot{q}$ can be written as:

$$\dot{q} = J^{\dagger}\dot{t} + (I - J^{\dagger}J)\dot{q_0},$$

where $\dot{t}$ is a task space velocity vector representing the desired movement of an end-effector. Also, J and $J^{\dagger}$ represent the Jacobian and the Jacobian pseudoinverse, respectively.

The second term $(I - J^{\dagger}J)$ represents an orthogonal projection matrix in the null-space of J and $\dot{q_0}$ is an arbitrary configuration space velocity; therefore different configuration space velocities can be obtained based on a null-space velocity, while fixing the same task space velocity. The null-space velocity is used for solving redundancy resolution via local optimization of additional motion objectives by defining a proper criterion such as joint limit avoidance, workspace centering [SK05], collision avoidance with obstacles [Kha86], or a combination of multiple criteria [BHG10].

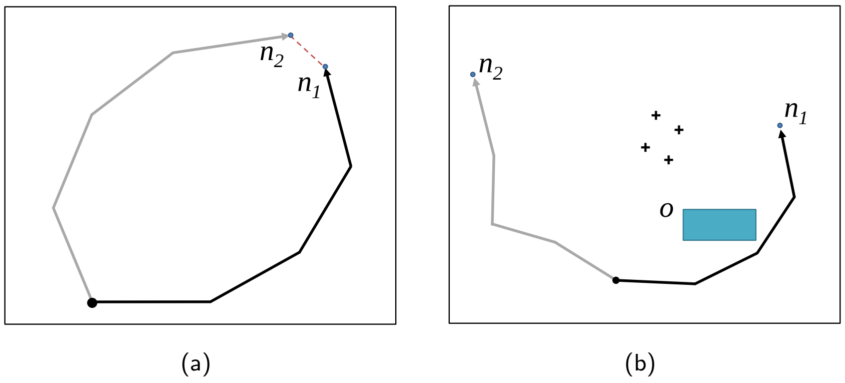


Figure 4.1: *(a) shows two different configurations of the manipulator, whose root is represented as the black dot and end-effector as an arrow. Their distance in the configuration space is much farther than the distance in the task space (represented as the dotted line) defined as the distance of end-effector position. (b) shows an inefficiency of the task space distance function to the sampling-based planner. The planner chooses n_1 as the nearest neighbor to random samples (plus marks), not n_2, while extensions from n_1 to these random samples are unlikely to succeed because of a nearby obstacle o shown as the square box.*

4.2.3 Motivation

TS-RRT directly grows its random tree in the task space with a distance function defined also in the task space (e.g., L_2 distance function when the task space is defined for a position of an end-effector). It is a common practice to use the distance function defined in its sampling space, the task space for the TS-RRT.

This simple approach, however, does not consider behaviors of local planners considering the inverse kinematics, as we discussed in Sec. 4.2.2. As a result, a task space distance function frequently leads to unsuccessful local planning, because a pair of nodes has a close distance in the task space, but a far distance in the configuration space; see its example shown in Fig. 4.1-(a). Such unsuccessful local planning causes inefficient performance, since its operations include inverse kinematics computation and several collision detection calls.

Fig. 4.1-(b) shows an example, where a planner tries to connect random samples shown in plus marks to a node n_1, which is selected as the nearest neighbor to these samples by the L_2 distance function defined on the task space. Extension trials from n_1 to these random samples (i.e., plus marks) tend to fail, because a nearby obstacle o restricts the movement of a manipulator. On the other hand, the connection would very likely succeed if extensions are tried from n_2 to those plus marks. This example shows that the task space distance function can be ineffective, since it does not consider behaviors of the local planner and obstacles.

The planner can be stuck into local minima, where further extensions do not lead to reach the goal, unless extensions from the tree explore around obstacles constraining the manipulator. Especially in environments with many obstacles, local minima arise very frequently and result in many unsuccessful local planning operations and collision detection checks. This problem becomes more pronounced with hyper-redundant manipulators, because of its high dimensionality.

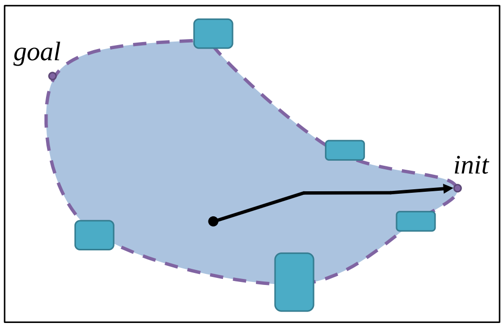


Figure 4.2: *An example of an ideal productive region given init and goal states. Boxes represent obstacles and the black dot represents the root of the manipulator. Sampling only in shaded regions is sufficient and efficient to move an end-effector (the black arrow) from init to goal.*

4.3 Sampling bias to Productive Regions

In this section we explain our sampling bias method for productive regions towards a goal state. We first introduce the notion of productive regions and utilize a maximum reachable area (MRA) to identify productive regions. Specifically we propose two sampling domains: a promising MRA and a detour sampling domain, the union of which defines our productive regions. We then give an overview of our algorithm, followed by a detail of how to compute the detour sampling domain and the MRA.

4.3.1 Productive Regions-Oriented Sampling Heuristic

In order to efficiently reach to the goal state, we define *productive regions*, a set of task space states that have a high probability of leading the random tree to the goal state. Fig. 4.2 shows a conceptual example of the productive regions given initial and goal states. We then bias our sampling distribution such that random samples for exploration are more frequently generated on these regions, instead of sampling the whole task space uniformly.

Correctly identifying those regions itself, unfortunately, is a very hard problem, and its complexity is the same to that of the motion planning problem, because it requires full analysis of the whole environment and the movement of the manipulator in the high-dimensional configuration space. Instead, we approximately identify such productive regions considering only local geometry.

In order to bias our sampling on productive regions, we use a concept of MRA for each node in the random tree. An MRA of a node is defined as a group of task space states where a manipulator can reach from the node by an employed local planner without having collisions. The MRA serves as an upper bound of a region in the task space where the node can be extended to, given a local planner and nearby obstacles around the node.

Fig. 4.3 shows a conceptual example of an MRA. Gray lines represent examples of manipulator configurations that can be extended from a configuration n by the local planner without collisions. The node n cannot be directly extended to g, because a obstacle o constrains the movement of the manipulator. The planner should generate a path by detouring the obstacle o in order to reach g. Fig. 4.6 shows a diagram on how to construct an MRA for the node n. We will explain how to compute the MRA in Sec. 4.3.3. In this section we give our overall algorithm utilizing MRAs.

We classify the MRA of a node to be *promising* or *unpromising*. We call the MRA promising, when the goal state is included in the area. Other MRAs are called unpromising. When the MRA of a node

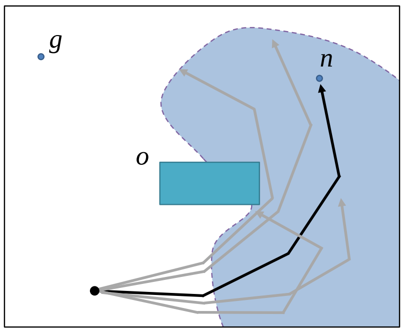


Figure 4.3: *A node n represents the end-effector of the manipulator (shown as line segments). Gray lines represent positions that the manipulator can reach by a local planner. On the other hand, the node n cannot reach to another node g by running a simple local planner because of a nearby obstacle o.*

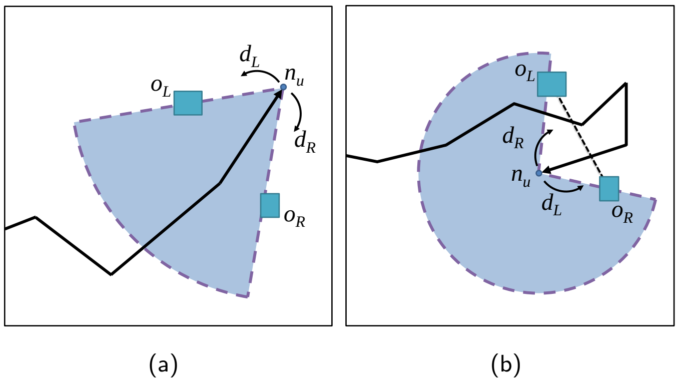


Figure 4.4: *(a) This figure shows a detour sampling domain for an unpromising node n_u in the $\mathbb{R}^2$ task space. (b) A detour sampling domain in $\mathbb{R}^2$ task space used for preventing to enter the local minima. In this case the manipulator passes multiple times through the virtual line (shown in the dotted black line) between two obstacles.*

contains the goal state, exploration in that area is clearly productive for reaching the goal, since we can easily reach the goal state by using the local planner with those samples generated for exploration. As a result, the union of all the promising MRAs are a subset of approximated, productive regions; detour sampling domains are another subset of approximated, productive regions.

The random tree, however, tends not to have any promising MRAs during iterations, especially in its early stage, when the exploration has not been spread much. Sampling an unpromising MRA of a node is unlikely to lead the random tree to reach the goal state. Consequently, it is more desirable to bias our sampling in a way that we detour from the unpromising MRA constrained by obstacles, and extend towards other potentially promising regions. We therefore propose to use a simple detour sampling domain for an unpromising node, in order to detour obstacles constraining the manipulator. Fig. 4.4 shows an example of a detour sampling domain for an unpromising node n_u.

In order to generate samples in approximated productive regions, we first generate a sample from the task space in a uniform manner, and confine it to our productive regions, which are approximated by the union of promising MRAs and detour sampling domains. The resulting distribution becomes uniform in the approximated productive regions. The pseudocode of our algorithm is shown in Algorithm 7.

By exploring promising MRAs and detour sampling domains for unpromising nodes that approxi-

Algorithm 7 PROT-RRT

```
Require: tree $T$, $q_{init}$, $x_{goal}$
  $x_{init} \leftarrow$ ForwardKinematics($q_{init}$)
  [$x_{init}$, $q_{init}$].$MRA \leftarrow$ ComputeMRA()
  $T$.AddVertex ([$x_{init}$, $q_{init}$])
  repeat
    repeat
      $x_r \leftarrow$ RandomStateInTaskSpace()
      [$x_n$, $q_n$] $\leftarrow$ NearestNeighbor($x_r$, $T$)
    until inProductiveRegion($x_r$, [$x_n$, $q_n$])
    $u \leftarrow$ Control([$x_n$, $q_n$], $x_r$)
    $q_{new} \leftarrow$ NewState($q_n$, $u$)
    if CollisionFree($q_{new}$, $q_n$) then
      $x_{new} \leftarrow$ ForwardKinematics($q_{new}$)
      [$x_{new}$, $q_{new}$].$MRA \leftarrow$ ComputeMRA()
      $T$.AddVertex([$x_{new}$, $q_{new}$])
      $T$.AddEdge([$x_n$, $q_n$], [$x_{new}$, $q_{new}$])
    end if
  until a collision-free path between $x_{init}$ and $x_{goal}$ is found
```

mate productive regions, we efficiently reach the goal state even for scenes with complex configurations of obstacles.

4.3.2 Computing Detour Sampling Domains

For an unpromising node, we first identify principal movement directions of an end-effector based on instantaneous directions computed by changing its joint values. In $\mathbb{R}^2$ space, principal movement directions of an end-effector are turning the end-effector into either its left or right directions, denoted as d_L and d_R, respectively (Fig. 4.4-(a)). We then compute an obstacle most constraining the manipulator in each movement direction. Fig. 4.4-(a) shows an example of our detour sampling domain of an unpromising node n in $\mathbb{R}^2$ task space. Constraining obstacles in directions of d_L and d_R are noted as o_L and o_R, respectively.

Computation of constraining obstacles to a certain direction can be implemented in many different ways. For example, we can compute the nearest obstacle overlapped with the span of the manipulator's movement to the given direction (d_L or d_R). Exact but expensive computation, fortunately, is unnecessary, mainly because we probabilistically bias our sampling based on computed obstacles. Among many alternatives, we chose to identify them by a simple method considering local geometry only. For each movement direction, we identify the nearest neighbor from the manipulator along the direction (e.g., d_L or d_R). When the nearest neighbor is located within a certain distance d from the end-link of the manipulator, we define it as the constraining obstacle in that direction.

We assume that obstacles farther than d do not constrain the local movement of the manipulator. When obstacles are not found in distance d for a node, we use uniform sampling around the node instead. The distance d can be chosen depending on environments and manipulator's configurations. We set d to be 40% of the maximum distance of any two points in the task space. Fig. 4.5 shows an example

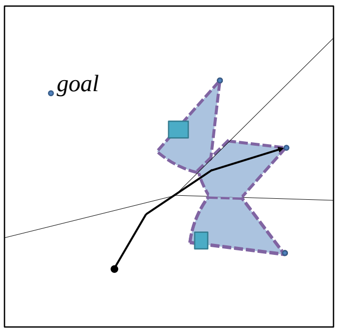


Figure 4.5: *In this case the manipulator should detour two obstacles (boxes) to reach the goal state. Shaded regions indicate detour sampling domains of nodes (blue dots) clipped by their Voronoi regions (black lines).*

of detour sampling domains for a set of unpromising nodes on regions constrained by nearby obstacles. Note that actual detour sampling domains are also clipped by the Voronoi regions of nodes, since each detour sampling domain is used when its node is chosen as the nearest node to a random sample.

An underlying assumption made so far for the detour sampling domain is that the manipulator passes only a single time through a virtual line drawn between two constraining obstacles o_L and o_R. There is, however, a case when the end-effector passes through the virtual line multiple times as shown in Fig. 4.4-(b). This kind of complication arises mainly because of the redundancy of the manipulator. In this case the sampling domain should be generated in an opposite direction, to prevent the manipulator re-entering the virtual line between two constraining obstacles. It can be distinguished simply by counting the number of links that pass through the virtual line between the constraining obstacles. When the number is even then the detour sampling domain is generated in an opposite direction.

4.3.3 Approximating the MRA

The exact computation of an MRA for each node is also challenging, because it requires to predict the movement of the high-dimensional manipulator by the local planner. Furthermore considering all those possible movements is an expensive computational operation. To reduce its complexity, we propose a simple approximation method for constructing the MRA. Our approximation considers only local geometry around the manipulator, instead of using global geometry information.

We identify obstacles that constrain the movement of the manipulator's end-effector. Specifically, we identify the nearest obstacles from a node in the same manner for computing constraining obstacles used for defining the detour sampling domain. Fig. 4.6 shows a schematic view on how to approximate the MRA for a node n. o_L and o_R represent two identified obstacles and the red line represents a base line for detouring in $\mathbb{R}^2$ task space. Moving the end-effector below the line is regarded as detouring from a region constrained by those two obstacles, and thus the regions below the line is not included in the MRA.

The MRA is approximately constructed by one circular sector (a), and two half-circles (b) and (c), shown in Fig. 4.6. The circular sector (a) represents an area where the end-effector can reach by stretching its joints. We thus compute the circular sector with its center located at the root of the manipulator and its radius is the length of the manipulator. Each side of the circular sector is computed by either one of two obstacles. One left half-circle (b) represents an area where the manipulator can reach with a physical contact between the left obstacle o_L and the manipulator. Its radius is set to the

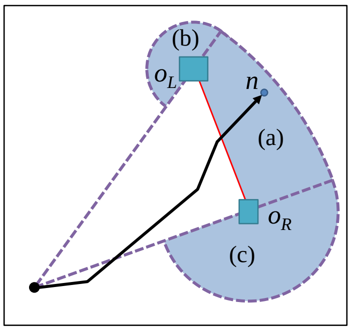


Figure 4.6: *Approximately constructed Maximum Reachable Area (MRA)*

manipulator's length minus the distance between the root and the obstacle. In a similar manner the right half-circle (c) is computed with the right obstacle o_R.

The overhead of computing approximated MRA is relatively very small, because the approximated MRA for each node is constructed with only few obstacles and simple geometric operations. In addition, this computation is performed only once for each node, when a node is newly added to the random tree.

4.4 Experimental Results

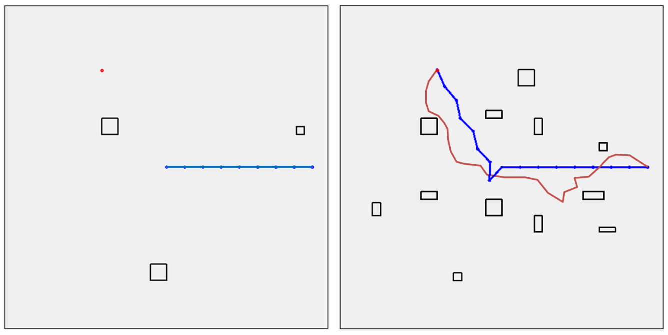
We implemented our method, PROT, and performed various experiments on an Intel i7 desktop machine that has 3.6GHz CPU. Additionally, we compared PROT to the standard RRT and TS-RRT against a planning problem for a serial rigid link manipulator, which has n number of revolute joints. A given initial pose is defined in the configuration space as a rest pose, and a goal end-effector position is defined in $\mathbb{R}^2$ task space. Any goal configurations are acceptable, when the end-effector locates at the goal position in the task space. For simplicity, we assume that the problem does not involve dynamics, and joints are unconstrained.

Constructed MRA and detour sampling domains for our productive regions are approximate by using only local geometry. This inaccuracy can aggravate the completeness of sampling-based planners. In order to avoid this issue and guarantee the probabilistic completeness for our method, we set our planner to enable our sampling bias with probability α and uniform sampling with probability $1 - \alpha$. We set α to 0.8 in our experiments.

4.4.1 Results and Comparisons

We tested PROT against two different benchmarks: one has a few number of obstacles and thus is relatively easy to solve, and another has complex configurations of obstacles that lead to local minima regions, which make the problem difficult. We ran each test 100 times and report the mean of those results. For our experiments, we perform collision detection in a brute-force way checking all obstacles with every link of the manipulator. We can also use a more advanced and efficient collision detection technique [KHH$^+$09]. Nonetheless our current collision detection unit takes a negligible cost.

Easy benchmark. We tested 8- and 20-dofs manipulators in an easy environment (Fig. 4.7-(a)) with three obstacles that do not interfere with the movement of the manipulator to reach goal state (red dot). The standard RRT takes much less time per node (about 0.94ms) than PROT (about 4.51ms) for the 20-dofs manipulator. The standard RRT quickly spreads the random tree in the configuration



(a) Easy benchmark (b) Difficult benchmark

Figure 4.7: *Tested environments with a 8-dofs manipulator. The initial pose is shown in blue, and the goal state is shown in red. (a) An easy benchmark contains three obstacles (black rectangles) which do not interfere with the movement of the manipulator to reach goal state. (b) This benchmark contains many obstacles (black rectangles) and gives rise to many potential local minima. A solution trajectory of the end-effector is shown in red.*

space, because of its cheap cost of generating a node. On the other hand, the local planner of PROT consisting of the Jacobian pseudoinverse and null-space calculations is expensive, leading to a higher cost for each node. Nonetheless, for the 8-dofs manipulator, the standard RRT does not give a solution within 60 s, while PROT finds a solution in 0.818 s on average. This is mainly because the sample process of PROT is performed in the task space, more efficiently leading to reach the goal state. The performance improvements go higher as the dimensionality of the configuration space goes up. For the 20-dofs manipulator, PROT takes 2.436 s on average, while the standard RRT fails to solve the problem in 10 minutes. In the literature of motion planning, it is well known that the complexity of planning grows exponentially as the dimension grows. This phenomenon is also known as the curse of dimensionality. On the other hand, PROT avoids this problem by performing the sampling process in the task space, which is in $\mathbb{R}^2$ for the tested benchmark.

Compared with TS-RRT that also performs the sampling process in the task space, our planner does not show considerable improvements in this simple benchmark. This is mainly because obstacles do not cause any local minimum cases or constrain the manipulator's movement. As a result, the union of productive regions in our method are almost same to the whole task space. Note that this is the worst case for our approach that emphasizes the overhead of our method. The overhead of our method includes computing MRAs and performing bias sampling, but it is small since it deals with only local geometry. Overall, our method showed a minor degradation, 12%, over TS-RRT for the 8-dofs manipulator and showed a slight improvement, 3% for the 20-dofs manipulator (Table 4.1). These results acquired from the simple benchmark demonstrate the low overhead of our method even compared to TS-RRT.

Difficult benchmark. We also compared different planners on a more challenging benchmark, which has many obstacles and gives rise to many potential local optima. This benchmark is shown in Fig. 4.7-(b) with 8-dofs manipulator. In this benchmark the standard RRT was unable to compute a solution in 300 s. As a result, we did not test 20-dofs manipulator for the standard RRT. In this difficult benchmark, productive regions are much smaller than the whole task space, because of many obstacles in the environment. For 8-dofs manipulator, PROT shows 1.87 times improvement over TS-RRT with

Table 4.1: *Experiment results for the simple benchmark*

8-dofs	RRT	TS-RRT	PROT
# of iterations	-	571.3	603.5
# of nodes	>53698	491.6	507.6
time (s)	>60	0.723	0.818
20-dofs	RRT	TS-RRT	PROT
# of iterations	-	708.7	666.8
# of nodes	>638295	510.9	540.1
time (s)	>600	2.519	2.436

Table 4.2: *Experiment results for the difficult benchmark*

8-dofs	RRT	TS-RRT	PROT
# of iterations	-	1342.2	529.9
# of nodes	>789120	661.9	401.7
time (s)	>300	13.43	7.19
20-dofs	RRT	TS-RRT	PROT
# of iterations	-	1832.0	723.9
# of nodes	-	690.4	417.6
time (s)	-	66.94	18.89

a uniform distribution (Table 4.2). This result demonstrates that our planner with biased sampling can explore the task space more efficiently and find a solution much faster than the planner with the uniform distribution. Furthermore, our method shows 3.54 times improvement for the 20-dofs manipulator. The improvement increases as we have a higher dimensional space, because the computation for one iteration consisting of local planning and several collision detections, takes higher portions of the overall computation time with a higher dimensional space.

4.5 Discussion

According to our experimentation, the computation of the Jacobian pseudoinverse local planner occupies a large portion (about 70%) of the total running time, while the portion of collision detections is relatively small. The main benefit of our biased sampling distribution is to reduce the number of expensive local planning trials for expanding the random tree.

Our method shows its strength over TS-RRT, especially for handling complicated environments, where many obstacles exist such that the movement of the manipulator is highly restricted by those obstacles and thus the random tree is likely to be stuck in local minima regions. In Table 4.2, the numbers of iterations and nodes represent to the total number of trials and their successes of local planning, respectively. TS-RRT generates more nodes in the random tree than our planner to solve the

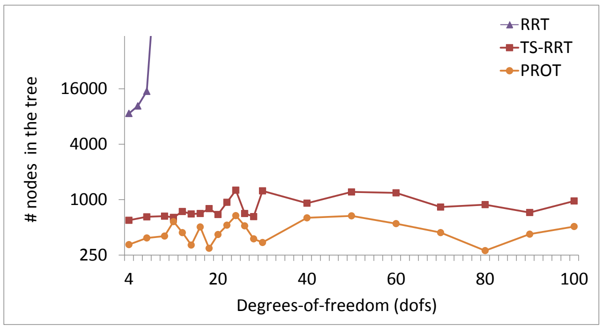


Figure 4.8: *The graph shows the total number of generated nodes in the tree after a first solution is found. Values in the Y-axis is on a log scale.*

problem, which implies that the TS-RRT spends more time for generating ineffective nodes (e.g. in local minimum) to find a solution.

Scalability. PROT can efficiently compute a solution for high dofs manipulators, even for more than fifty dimensions. This is because the dimension of the task space where the sampling process is performed is independent from the dimensionality of manipulators. Fig. 4.8 shows a graph showing the number of nodes of the random tree generated after finding a solution, as a function of dimensions of manipulators with different methods. This experiment was conducted in the difficult benchmark. Although the number of nodes required for finding a solution grows super-linearly for the standard RRT, our method uses a much smaller number of nodes, which are within an almost fixed range across different dofs tested up to a 100-dofs manipulator. Overall TS-RRT has a similar tendency but the total number of nodes is larger. Furthermore, our method shows faster performance than TS-RRT across all of tested dofs. This result is attributed to our productive sampling regions that effectively avoid the local minima. The scalability of our method could lend itself to other problems such as high dimensional snake-arm robots or continuum manipulators that might be used for diverse fields including medical surgery, collapse search and rescue, and inspection [Buc02]

4.6 Application to Cabled Mobile Robots

We have applied our method to a cabled mobile robot problem whose goal is to find a trajectory of a mobile robot that is connected to a fixed point by an under-actuated constant length cable. This problem is difficult, since the cable imposes constraints on the movement of the mobile robot: the cable can collide with obstacles and make the robot moving no further than the cable length. Fig. 4.9 shows an example where the cabled robot should detour obstacles constraining the cable to reach the goal position, because the direct movement to the goal is restricted by the contacts between the cable and obstacles. The motion planning problem that the cabled mobile robot faces is essentially same to that of our redundant manipulators. The difference is that the cable cannot be actuated directly but follows the mobile robot's movement by respecting only the kinematics of links representing the cable.

We approximate the cabled robot problem by the manipulator planning problem. Specifically we treat a cabled robot as a redundant manipulator that has an infinite number of dimensions. In practice, we have found that one hundred dimensions are enough to approximate the movement of the real cable. The end-effector of our manipulator then represents the position of the mobile robot. In other words,

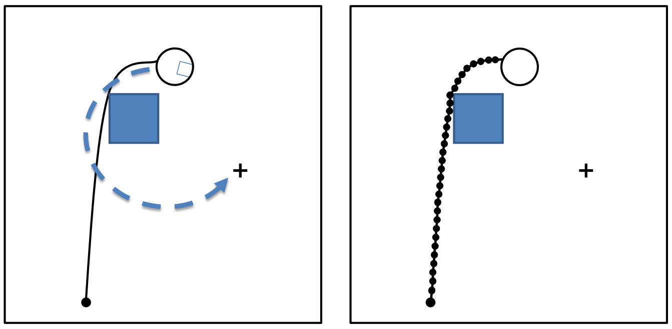


Figure 4.9: *(left) An example where the cabled mobile robot should detour the obstacle (squared box) to reach to the goal position (the plus mark). (right) An approximated 33-dofs manipulator from the cabled mobile robot. The end-effector is represented by the same-sized circle as the mobile robot, and dots represent to joints of the manipulator.*

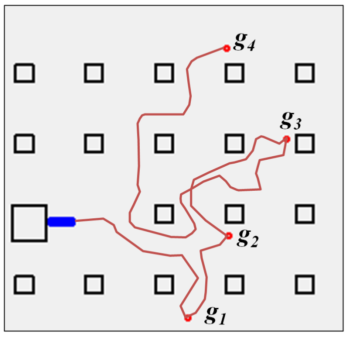


Figure 4.10: *This figure shows a test environment for a cabled mobile robot. The robot should reach goal positions (red dots) sequentially by avoiding obstacles (squared boxes). The start position is located near the large box and the robot's cable (blue lines) is approximated by a 100-dofs manipulator. The cable is piled initially. The computed solution trajectory is shown in the red curve. The end-effector is moved back to reach g_4 from g_3.*

the trajectory of the manipulator's end-effector acts as a feasible path for the mobile robot that guides the cable to move around obstacles. We let the cabled mobile robot to follow the solution trajectory of the manipulator's end-effector.

Fig. 4.10 and Table 4.3 show a test environment used for the cabled mobile robot and its results, respectively. In the test the robot starts from its base and visits four goal positions sequentially. We approximate the cabled robot with a 100-dofs manipulator. We compared our method and TS-RRT; we ran 100 times for each algorithm with a maximum running time of 800 seconds. PROT shows 3.61 times faster performance than TS-RRT. We also measure a success ratio of finding a solution given the maximum running time out of 100 tests. Our planner's success ratio, 80%, is much higher than that, 35%, of TS-RRT. These results demonstrate advantages of our method over TS-RRT in the tested application.

Table 4.3: *Experiment results for the cabled mobile robot benchmark*

	TS-RRT	PROT
# of iterations	6563.9	2752.8
# of nodes	1300.5	527.3
time (s)	250.46	69.24
Success ratio	35%	80%

Chapter 5. RRT planner with motion database for continuous kinodynamic motion planning problem with complex dynamics

5.1 Overview

In this chapter, we propose a novel kinodynamic motion planner, which is designed for complex dynamics environments with many obstacles. Our method builds motion database as a preprocessing in order to eliminate duplicated simulation and increase the efficiency of the planner (Sec. 5.3.1). We then propose efficient tree extension method using motion database (Sec. 5.3.2) and propose validating conditions to efficiently update the interpolated motions during iterations (Sec. 5.3.3). Constructed motion database can be used with our planning method for other problems such as different composition of obstacles or different start/goal state.

5.2 Background

5.2.1 Problem Definition

In the kinodynamic motion planning, the state space is used for the same purpose as the configuration space for the holonomic planning problem, which represents a transformation that could be applied to a geometric model of the robot. The state space for the kinodynamic planning is defined as $x = (q, \dot{q})$, where q denotes a state in the configuration space. The differential or non-holonomic constraints can be described by a forward propagation function $f : X \times U \rightarrow \dot{X}$, where X is the state space and U represents a set of allowable controls, or the control space. A solution path to the planning problem consists of a sequence of controls and its durations, and states which can be obtained sequentially by the integration of propagation function f.

In order to integrate the propagation function f, firstly a set of motion equations should be provided and the dynamics in-between environments and a robot, and the property of the robot also should be derived. For many problems, the propagation function might not be derived because of its complexity, or the function is provided with simplified and approximated form that causes inaccuracy of the planning environments. In order to provide more generality and accuracy, a complicated physics-based simulation engine, which works as a black box to the motion planner, is used for generating motions instead of the propagation function integration [SK12], although the physics-based simulation is more computationally expensive than the integration of propagation function.

5.2.2 Physics-based simulation engine

Many libraries are provided for performing physics-based simulation. These libraries usually assume all objects in environments are rigid and apply the numerical integration of the differential equations. In physics-based simulator, a simulation duration is discretized and the propagation function is evaluated for this duration only. This discretization interval, called a simulation step, normally does not change during whole simulation. Simulating for specific duration is achieved by executing simulation for several numbers of single simulation step. The simulation step has a trade-off between the quality of solution paths and the running time of the planning. The typical simulator setting fix the simulation step all time and we use default value as suggested by the simulator community. We use the Open Dynamic Engine [ope] for the current implementation among well known libraries such as PhysX, Bullet, Vortex etc. Please note that our planner is not restricted to specific library. Any library can be coupled to our method.

5.2.3 Kinodynamic RRT planner

Kinodynamic RRT planner is derived from the RRT for holonomic planning [LK01]. Given a random tree which is initialized from a start state, it incrementally expands the random tree by randomly choosing an existing node and a control. In order to satisfy the principal property of RRT probabilistic completeness from rapid exploration of the space, selection of an existing node is conducted by finding the nearest neighbor in the random tree, x_n of a random state, x_r in the state space. Then a control u is selected as the one that pulls the node x_n toward the random state x_r. In order to calculate the control that satisfies this condition, integration of the propagation function is required. In this chapter we target to solve the problem with complex dynamics, and for that reason the propagation of motion in time should be executed by the physics-based simulator. In order to find the control u satisfying requirement that pulling the node x_n toward the random state x_r among all available controls, we compute the best control u_{best} after trying all available controls and choosing the one which extends the state x_n as close as possible to the x_r. In practice, searching to the best control is achieved iteratively in which the finite number of controls are generated randomly and the best one is selected. We call this approach as a n-control Kinodynamic RRT, where n denotes the fixed number of trials to find the best control. Finally, a new state x_{new} is generated by taking u_{best} from x_n, and its control u_{best} are added to the random tree if there is no collision in between x_n and x_{new}. Its pseudocode is shown in Algorithm 8.

Algorithm 8 n-control Kinodynamic RRT

```
Require: tree $T$, $n$
  $T$.AddVertex ($x_{init}$)
  repeat
    $x_r \leftarrow$ RandomStateInStateSpace()
    $x_n \leftarrow$ NearestNeighbor($x_r$, $T$)
    $u_{best} \leftarrow$ FindBestControl($x_n$, $x_r$, $n$)
    $x_{new} \leftarrow$ Extend($x_n$, $u_{best}$)
    if CollisionFree($x_{new}$, $x_n$) then
      $T$.AddVertex($x_{new}$)
      $T$.AddEdge($x_n$, $x_{new}$, $u_{best}$)
    end if
  until a collision-free path between $x_{init}$ and $x_{goal}$ is found
```

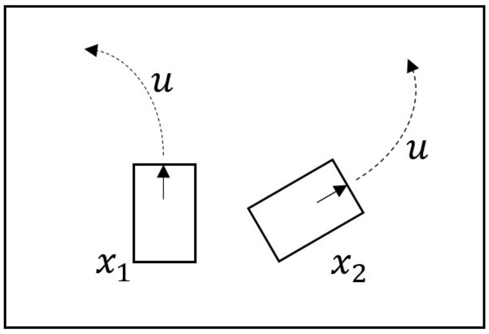


Figure 5.1: *Motivational example*

5.2.4 Motivation

The kinodynamic RRT can solve the problem if iterations go to infinity [LK01]. The running time for each iteration is decomposed into propagation, nearest neighbor search and other parts such as collision detection, tree expanding, etc. When simulator is used for propagation process, the executing simulations take the highest portion of the running time (about 75%, See Table 5.1)

For an iteration of RRT, the simulator is executed several times for computing controls given source and target states. We found that many duplicated simulation occurs during overall iterations when environments consist of objects which have similar dynamics property such as the friction constant. The number of duplication becomes especially large from the perspective of simulator where the robot's property in global coordinate is often ineffective to the simulation results. For example, when a mobile robot moves around on the infinite floor which have same material property, the dynamics of the robot in the environment are same for all spaces, therefore the simulations are identical for states of which properties in robot's local coordinate are same. In Fig. 5.1, two stationary robots have distinct state: position and orientation are different. However, the properties in local coordinates such as velocity, are identical. We can use same simulation results for both states instead of executing simulations twice, if the same control is applied to both states such as applying force toward heading direction.

Eliminating duplicated simulations can decrease the overall performance of the kinodynamic planning problem because of its high portion in the total running time. Furthermore, several simulations are precomputed and constructed as the database, then the overall efficiency of the planner in run-time can be improved much. The constructed database also can be used for other problems sharing same dynamics model such as different composition of obstacles and different initial/goal states.

5.3 Algorithm

In this section we explain our kinodynamic sampling-based planner with precomputed motion database. Our planner is based on the RRT-based kinodynamic planner [LK01] which have a similar manner of exploration in the state space. We first show the structure of proposed motion database and how to construct it. We then explain our extension algorithm with motion database and how to validate interpolated motions during planning. Finally, we give an overall planning algorithm.

5.3.1 Building motion database

For constructing simulation database, we define *the local state space* that is projected space from the global state space used for planning process. The local state space is a set of states which can be described in robot's local coordinate such as velocities, local geometric configuration such as joint angle for an articulated robot. The defining projections could be done automatically by analyzing state space components or specified by the user. In most case, the specification can be done easily because the classification whether robot's component is embraced in local or global coordinate is intuitive. After the local state space is defined, proper converting function $f_c : X \to X_{local}$ should be defined, where X is the original state space, or global state space and X_{local} denotes the local state space. The converting can be easily defined by several geometric transformations.

An input of a motion consists of a source state and a control to execute. And results of simulation are stored in the motion as a trajectory of the robot and a final state after propagation. Every states of motion is represented in the local state space and the trajectory is generated as a set of relative geometric configurations of the robot from an initial position to a final position. When a motion is applied to a state x_{source} in the global state space, the trajectory and the final state of the motion are converted in the global state space by using the information of x.

In order to build the motion database, motions can be generated randomly or provided from the user. Richer database consisting of motions which is more useful to solve the problem can be generated by carefully analyzing environments and a robot, we use the random generation in current experiments. We randomly generate a source state, a control, and control duration, then store results of simulation for a fixed amount of time. We maintain the data as the lookup table indexed by a state in the local state space. Constructed database return a set of motions from a given state by the nearest neighbor search that retrieves motions that the distance of given state and source state of motion are less then some threshold, t_{sim}. When the state of retrieved motion is not same as the input state, we interpolate a motion in order to fit the motion for the input state. t_{sim} determines an allowable similarity of states in the local state space.

5.3.2 Exploration with the motion database

Our planner is based on the n-control kinodynamic RRT planner (Sec. 5.2.3). For each iteration, we randomly generate a sample x_r in the global state space, and find its nearest neighbor node x_n in the random tree. We then compute the best control input u_{best} by retrieving a set of controls from the motion database that stores the precomputed propagation results by simulator (Sec. 5.3.1) instead of executing simulator. x_{new} is computed from the control and the state x_n, then a new node and edge are appended to the random tree if there is no-collision.

The number of retrieved controls for calculating the best control to extend from x_n varies depending on the precomputed motion database. More times are spent for building database, the number of retrieved controls from the database would increase. However, covering all the local state space is impossible and inefficient because the size of the state space grows exponentially as the dimension of the state space grows. Therefore, the number of retrieved controls for a given state would be less than n or zero at worst case. Searching the best control among only limited number of controls might bias the expansion of the tree in the global state space and interfere the rapid exploration of the state space, therefore affects the overall performance of the planner. In order to solve this inefficiency, we selectively use our control generation method as a probability function, $p(n_{found})$, where n_{found} represents to the

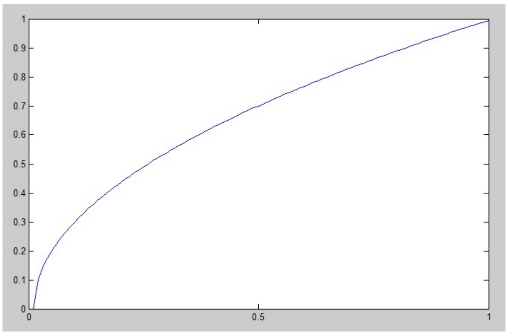


Figure 5.2: *A probability distribution function to decide whether the motion database is used.*

number of controls found from the database for a given state. We use the probability function as follows:

$$p(n_{found}) = (n_{found}/n)^{\alpha}$$

where n is the maximum number of control trials and α is a parameter in range of (0,1). Fig. 5.2 shows a plot of function when α is 0.5. We use a *FindBestControl(·)* (Sec. 5.2.3) with probability $1 - p(n_c)$.

5.3.3 Validating interpolated motions

The states computed by the motion database could be incorrect because interpolation is occurred for retrieving motions (Sec. 5.3.1). We define an interpolation error as the distance between a query state and a source state of retrieved motion. The interpolation error of a single extension by the motion database is at most t_{sim}, because t_{sim} defines the maximum allowable distance for interpolated motion (Sec. 5.3.1). Therefore, t_{sim} decides the quality of the path in our planning method. The interpolation error can be accumulated as the tree is expanded. If a newly appended motion is added to the node that already have an error, then the distance of interpolated motion and actual motion becomes large. The error likely becomes larger and larger as more nodes are appended by interpolated motion. For simplicity, we define an accumulated error of a node as the sum of the current error and parent node's error.

When a solution path is found, we validate every motions in the solution path using the simulator in order to make sure that it achieves the goal. If validated path does not satisfy the goal of the problem, we update validated motions in the random tree then continue iterations until new solution is found. When the validated motion is changed from the interpolated motion, collision detections are also performed again. If collision occurs for the validated motion, we remove the colliding edge and all nodes connected from the edge in the random tree.

The validation of motions affects the efficiency of the planner, because the validation requires a simulation and spends a lot of time. However, invalid interpolated motion that would cause collision in validation time could mislead exploration or incur a mass deletion of nodes in the tree, because all children nodes should be deleted if a parent node is disabled by collision. If t_{sim} is set small enough, a single motion interpolation does not affect the exploration of the state space in most case, therefore it is efficient to validate a solution path only.

However, the interpolation error could become problematic when error is accumulated much or many obstacles exist in environments. A node with high accumulated error could affect the exploration of RRT to be biased with a wrong information. Moreover, small interpolation error can cause the same negative effect when the tree is expanded near obstacle region, specially around narrow passages. Fig. 5.3 shows

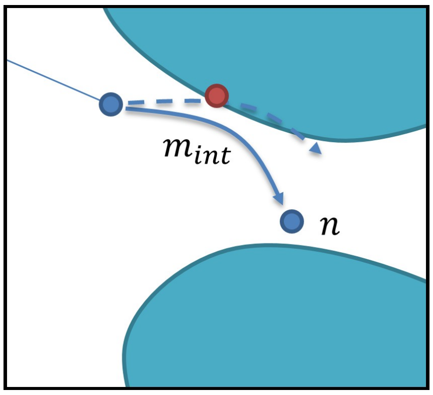


Figure 5.3: *This figure shows an example when an interpolated motion causes a collision with small error. The interpolated motion m successfully extends the tree toward a node n (a solid line), the real motion (a dotted line), however, causes a collision with obstacles.*

an example. The interpolated motion m successfully extends the tree toward a node n (a solid line). However, the real motion (a dotted line) causes a collision although the interpolation error of m is small.

In order to maintain the effectiveness of our planner, validations should be done during iterations occasionally considering accumulation error and obstacles around. We check three conditions for decision of validation when a new node is added to the random tree. If any of condition is satisfied, then we validate a path from the root to the newly inserted node. Three conditions are as follows: 1. if accumulated error of the node is larger than $threshold_a$, 2. if the distance from the node to the nearest obstacle are less than $threshold_o$, 3. if the distance from the node to the goal is smaller than $threshold_g$ For overall efficiency of the planner, the number of validations should be as small as possible. Therefore three parameters should be carefully chosen depending on the environments.

5.3.4 Overall algorithm

A pseudocode of the overall algorithm of our planner is shown at Algorithm 9.

5.3.5 Completeness issue

Our planning method can efficiently explore the state space postponing the time-consuming propagation simulation by using precomputed motion database. However, the lack of the number of retrieved controls and errors can degrade the exploration of the state space, although we provide a decision probability function (Sec. 5.3.2) and validation conditions. In order to guarantee the probabilistic completeness for our method, we set our planner to use our extension method with precomputed data with probability α and uniform sampling with probability $1 - \alpha$. We set α to 0.85 in our experiments.

5.4 Experimental Results

We implemented our method on an Intel i7 desktop machine that has 3.6GHz CPU and 16Gb main memory. Our method is built upon an Open Motion Planning Library (OMPL [SMK12]) and we use a Proximity Query Package (PQP) library [LGLM99] for the collision detection. We compared our method to the kinodynamic RRT with n control trials, called n-RRT (Sec. 5.2.3) against a kinodynamic planning problem for a rigid robot with dynamics.

Algorithm 9 n-control Kinodynamic RRT with motion database

```
Require: tree $T$ , $n$, motion database $db$
  $T$.AddVertex ($x_{init}$)
  repeat
    $x_r \leftarrow$ RandomStateInStateSpace()
    $x_n \leftarrow$ NearestNeighbor($x_r$, $T$)
    $controls \leftarrow$ RetrieveControls($db$, $x_n$)
    if with probability of $p$( size of $controls$ ) then
      $u_{best} \leftarrow$ FindBestControlAmong($x_n$, $x_r$, $controls$)
      $x_{new} \leftarrow$ ExtendFromDatabase($x_n$, $u_{best}$)
    else
      $u_{best} \leftarrow$ FindBestControl($x_n$, $x_r$, $n$)
      $x_{new} \leftarrow$ Extend($x_n$, $u_{best}$)
    end if
    if any of three validation conditions satisfied then
      ValidatePath ($x_{new}$)
    end if
    if CollisionFree($x_{new}$, $x_n$) then
      $T$.AddVertex($x_{new}$)
      $T$.AddEdge($x_n$, $x_{new}$, $u_{best}$)
    end if
  until a collision-free path between $x_{init}$ and $x_{goal}$ is found
```

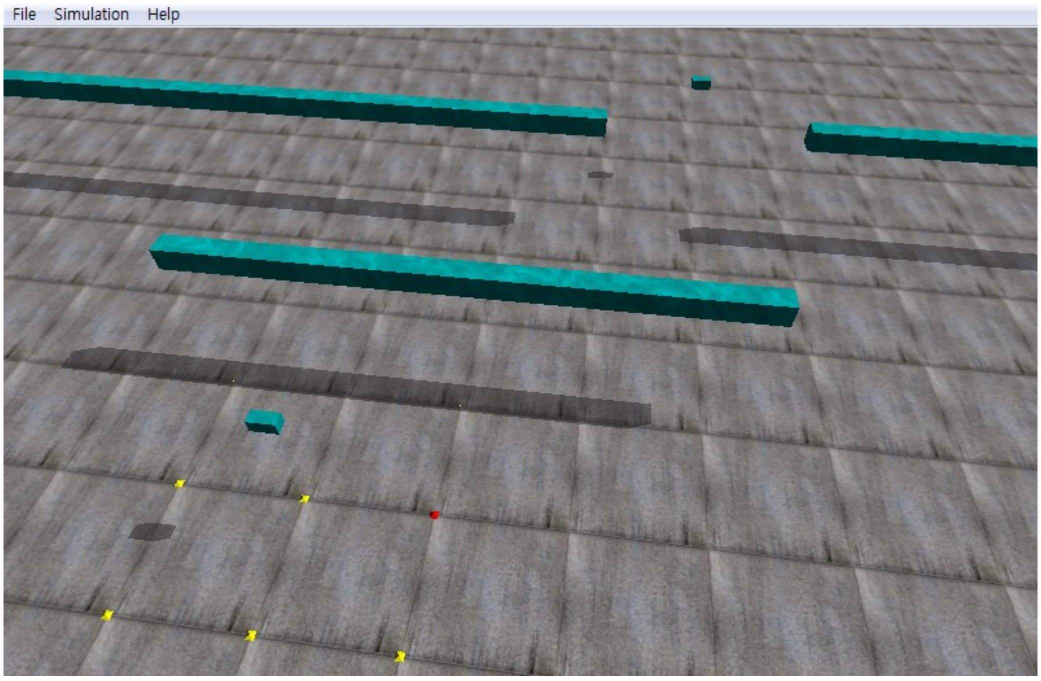


Figure 5.4: *Tested environment for the space shuttle robot. An initial state of the robot is in the bottom. The robot has to move around the obstacles and pass through the hole behind.*

5.4.1 Benchmark Model

We conducted experiments for a three dimensional space shuttle robot in a space environment where no gravity exists and many obstacles generate narrow passages. Fig. 5.4 shows our experiment environment.

In our model, the global state space has the following components of the robot:

$$\mathbf{x} = (\mathbf{p}, \theta, \mathbf{v}, \omega)$$

where $\mathbf{p}$ denotes global position in a three dimensional space (x, y, z), θ denotes an orientation which is represented as a unit quaternion in $SO(3)$ space, $\mathbf{v}$ denotes linear velocity $(v_x,\ v_y,\ v_z)$, and ω denotes angular velocity $(\omega_x,\ \omega_y,\ \omega_z)$. The local state space, which is used in the motion database, is defined as linear velocity and angular velocity in local coordinate space of a robot:

$$\mathbf{x}_{local} = (\mathbf{v}_{local}, \omega_{local})$$

where $\mathbf{v}_{local}, \omega_{local}$ represent to the converted linear and angular velocities into the robot's local coordinate space.

We use a simple metric for computing the distance between states based on a weighted Euclidean distance for all components of states. The equation of distance can be defined as:

$$d(\mathbf{x_1}, \mathbf{x_2}) = w_p(\|\mathbf{p}_1 - \mathbf{p}_2\|)^2 + w_o(1 - \mid \boldsymbol{\theta_1} \cdot \boldsymbol{\theta_2} \mid)^2 + w_v(\|\mathbf{v}_1 - \mathbf{v}_2\|)^2 + w_\omega(\|\omega_1 - \omega_2\|)^2$$

where w_p, w_o, w_v, w_ω are weights for each components.

Finally, the control space of the robot has three dimensions: force along to x-axis in robot's coordinate which represents to the forward and reverse engine of the space shuttle, pitch torque and yaw torque (See Fig. 5.5). Combining of three controls, the robot can reach to any place in the three-dimensional workspace with complex maneuverer.

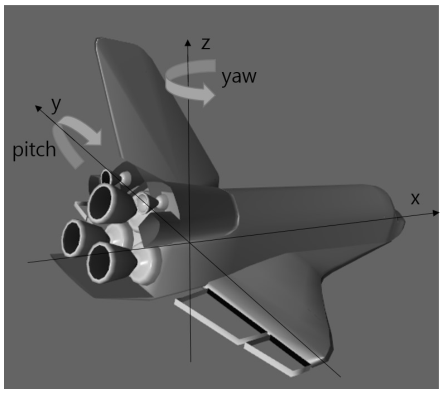


Figure 5.5: *Space shuttle model [NAS] and its control dimensions*

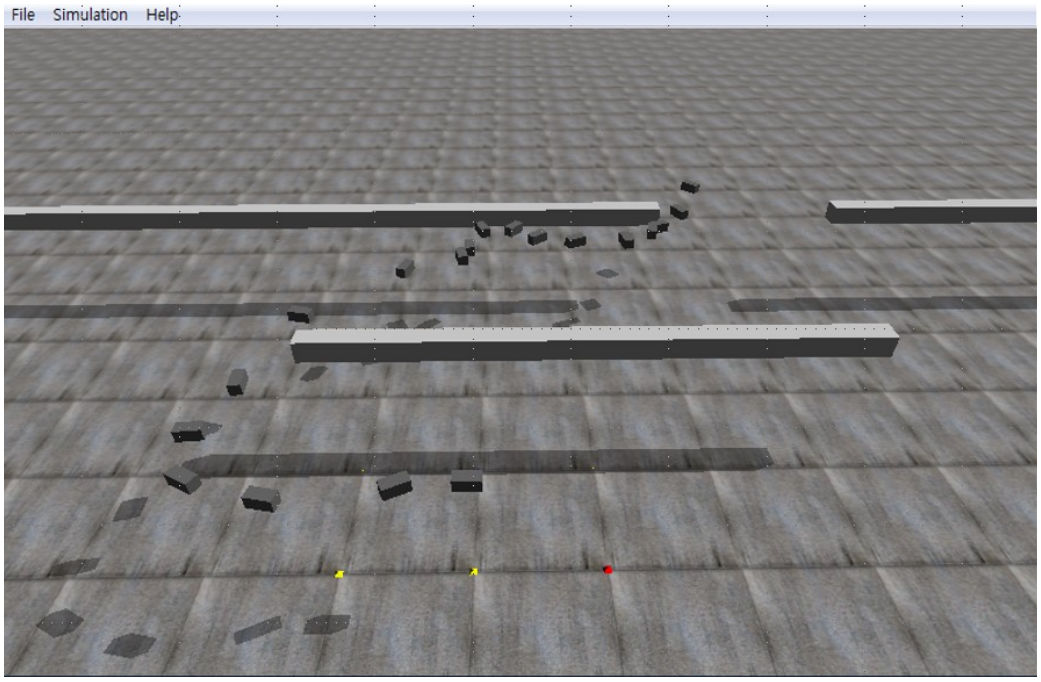


Figure 5.6: *Solution path of our benchmark*

5.4.2 Results and Comparisons

We tested two different experiments in which various times are used for preprocessing, therefore the number of generated motions are different. We ran the experiment 100 times and report the mean of those results. Table 5.1 shows experimental results for those two experiments and the kinodynamic RRT with 30 control trials. In first experiment (ours 1), we generate about 40,000 motions for the database, and about 120,000 motions are generated in second experiment (ours 2). Fig. 5.6 shows a one of solution path by our planner.

n-RRT consumes many portion of time for simulation queries (about 76%) to solve the problem, while ours uses smaller portion of time for simulation as more motions are stored in the motion database. Richer database decreases the portion of time for simulation, because many expensive simulation execution can be replaced to database retrieval which is relatively faster. ours2 is about 1.42 times faster than n-RRT, in which 30 s are used for preprocessing. Note that the generated motion database can be reused for different problems such as different composition of obstacles or initial, goal states.

Table 5.1: *Performance comparison table between the kinodynamic RRT with n controls (n-RRT, Sec. 5.2.3) and our method (ours, Sec. 5.3.4). In ours 1, 40,000 motions are generated and in ours 2, 120,000 motions are generated in preprocessing phase.*

	n-RRT	ours 1	ours 2
# of iterations	7693.88	13943.44	10484.78
# of nodes	311.10	437.08	281.48
Total time (s)	125.11	140.96	88.22
Simulation time (s)	94.61	50.53	25.51
% of simulation	75.62	35.85	28.92

Table 5.2 shows a breakdown of the running time of kinodynamic RRT with n controls and our method with the different amounts of preprocessing time, for different components including the simulation, data retrieval from the database, collision detections, and other parts (e.g., maintaining graph, validating nodes based on update rules in Sec. 5.3.3, etc.).

Table 5.2: *Breakdown of the running time*

	n-RRT	ours 1	ours 2
Total time (s)	125.11	140.96	88.22
Simulation time (s)	94.61	50.53	25.51
% of simulation	75.62	35.85	28.92
Retrieval time (s)	0	10.92	8.99
% of retrieval	0.00	7.74	10.19
Validation time (s)	0	8.17	11.91
% of validation	0.00	5.80	13.50
etc. (s)	30.50	71.34	41.81
% of etc.	24.38	50.61	47.39

Chapter 6. Conclusion

In this thesis, we propose novel sampling-based motion planning methods for different types of robot and constraints: a free-flying rigid robot (SR-RRT in Chap. 3) , a manipulator robot with kinematic constraints (PROT in Chap. 4), and the kinodynamic planning problem with complex dynamics (Chap. 5).

SR-RRT is designed for a wide variety of environments that have or do not have narrow passages. Instead of performing retraction operations exhaustively, we selectively perform retraction operations, only if samples to be retracted are deemed to be around narrow passages. To decide whether a sample is near a narrow passage, we propose a *bridge line- test*, an inexpensive filtering operation. We perform principal component analysis (PCA) and generate the line with a higher probability to cross narrow passages in order to achieve a high accuracy even in high dimensional configuration spaces. Also, in order to generate more random samples near narrow passages, we present a *non-colliding line-test* that detects wide-open free spaces. Our method shows the highest performance among all the tested methods with all the tested benchmarks, while the performance of other methods depend on the type of environments. This result demonstrates higher robustness and generality of our method.

For a hyper-redundant manipulator robot with kinematic constraints, we propose a novel sampling strategy, PROT, to improve the performance of a trajectory planner that directly explores the task space. We propose to use a concept of productive regions, which effectively guide a random tree towards a goal state. We then propose two sampling domains for each node in the random tree: a promising maximum reachable area (MRA) and a detour sampling domain. When the MRA of a node contains the goal state, we classify it as a promising node and use its MRA as its sampling domain. For other nodes that do not contain the goal state, we call them unpromising and additionally construct a detour sampling domain for detouring obstacles constraining the manipulator. Proposed sampling strategy shows up to 3.54 times improvements on recent works for a 20-dofs manipulator in a challenging benchmark including narrow passages. We have also applied our method to solve a cabled mobile robot problem by approximating the problem as a 100-dofs manipulator planning problem.

For planning under kinodynamic constraint, we propose a novel extension method with a precomputed robot motion database for sampling-based planners with complex dynamics. We replace expensive simulation executions with retrievals from the database which is defined in the local state space. In order to increase the ratio of retrieval, we employ motion interpolation. We also perform validations during iterations to reduce a bias exploration by interpolation errors. By using the motion database, many redundant simulation executions can be saved, therefore the overall performance of the planner is improved. Also, the database can be re-used for different composition of obstacles or different problem definitions.

There are many avenues for future research directions. For SR-RRT, we would like to design more accurate, yet efficient filtering methods. Also, we would like to identify collision- free paths in a multi-resolution approach [GFC09, RTPA06] in order to more efficiently find such paths. It will be very challenging to design a multi-resolution technique for environments that contain narrow passages. Our planner requires some parameters including ones inherited from the retraction operation. Eliminating the effort to tune the algorithm can be beneficial. Also, adopting GPU-based techniques can significantly accelerate the performance of our approach. Finally, the optimization-based retraction method has been extended to articulated robots [PZM10]. We would also like to extend and test our method to such cases. For PROT, we would like to compute productive regions in various task spaces. Currently, we have experimented $\mathbb{R}^2$ task space and thus the manipulator has only two principal movement directions. In order to extend the methods to $\mathbb{R}^3$ task space, a combinatorial analysis of the manipulator configuration is needed to identify its principle movement directions for the computation of the MRA and the detour sampling domain. We also would like to extend our method for various manipulators with prismatic joints or multiple end-effectors. Finally, we would like to combine asymptotically-optimal algorithms [PKS+11] within our method for finding the optimal solution. For the extension methods using the motion database, we would like to increase the efficiency of the motion database by analysis of a robot's movement. The analysis of movements can be achieved by automatically or manually. Specially, learning from human expert's demonstration which has been research for many years in robotics [BS02, SIB03, BCG06, YYN11], can build the database more useful for planning.

References

[ABD$^+$98] N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. Obprm: An obstacle-based prm for 3d workspaces. *Int'l. Workshop on the Algorithmic Foundations of Robotics*, pages 197–204, 1998.

[BB07] B. Burns and O. Brock. Single-query motion planning with utility-guided random trees. In *ICRA*, pages 3307–3312, 2007.

[BCG06] Aude G Billard, Sylvain Calinon, and Florent Guenter. Discriminative and adaptive imitation in uni-manual and bi-manual tasks. *Robotics and Autonomous Systems*, 54(5):370–384, 2006.

[BCLM06] M.S. Branicky, M.M. Curtiss, J. Levine, and S. Morgan. Sampling-based planning, control and verification of hybrid systems. *IEE Proceedings Control Theory and Applications*, pages 575 – 590, 2006.

[BDG85] James E Bobrow, Steven Dubowsky, and JS Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17, 1985.

[BHG10] Matthias Behnisch, Robert Haschke, and Michael Gienger. Task space motion planning using reactive control. In *IROS*, pages 5934–5940, 2010.

[BKDA06] Dominik Bertram, James Kuffner, Ruediger Dillmann, and Tamim Asfour. An integrated approach to inverse kinematics and path planning for redundant manipulators. In *ICRA*, pages 1874–1879, 2006.

[BOvdS99] V. Boor, M.H. Overmars, and A.F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *ICRA*, pages 1018–1023, 1999.

[BS02] Cynthia Breazeal and Brian Scassellati. Robots that imitate humans. *Trends in cognitive sciences*, 6(11):481–487, 2002.

[Buc02] Rob Buckingham. Snake arm robots. *Industrial Robot: An International Journal*, 29(3):242–245, 2002.

[BV02] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *IROS*, pages 2383–2388, 2002.

[CB93] Gregory S Chirikjian and Joel W Burdick. Design and experiments with a 30 dof robot. In *ICRA*, pages 113–119, 1993.

[CFL04] Peng Cheng, Emilio Frazzoli, and Steven M LaValle. Improving the performance of sampling-based planners by using a symmetry-exploiting gap reduction algorithm. In *ICRA*, volume 5, pages 4362–4368. IEEE, 2004.

[CJS07] J. Cortes, L. Jaillet, and T. Simeon. Molecular disassembly with rrt-like algorithms. In *ICRA*, pages 3301–3306, 2007.

[CL95] H. Chang and T. Li. Assembly maintainability study with motion planning. In *ICRA*, 1995.

[CL01] Peng Cheng and S.M. LaValle. Reducing metric sensitivity in randomized trajectory design. In *IROS*, volume 1, pages 43 –48 vol.1, 2001.

[DA11] Jory Denny and Nancy M. Amato. Toggle prm: Simultaneous mapping of c-free and c-obstacle - a study in 2d -. In *IROS*, pages 2632–2639, 2011.

[DL11] S Dalibard and J.P. Laumond. Linear dimensionality reduction in random motion planning. *Int'l. Journal of Robotics Research*, 2011.

[DRF+08a] Rosen Diankov, Nathan Ratliff, Dave Ferguson, Siddhartha Srinivasa, and James Kuffner. Bispace planning: Concurrent multi-space exploration. *Proceedings of Robotics: Science and Systems IV*, 63, 2008.

[DRF+08b] Rosen Diankov, Nathan Ratliff, Dave Ferguson, Siddhartha Srinivasa, and James Kuffner. Bispace planning: Concurrent multi-space exploration. *Proceedings of Robotics: Science and Systems IV*, 2008.

[DXCR93] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993.

[ERP+04] Deniz Erdogmus, Yadunandana N Rao, Hemanth Peddaneni, Anant Hegde, and Jose C Principe. Recursive principal components analysis using eigenvector matrix perturbation. *EURASIP Journal on Applied Signal Processing*, 2004:2034–2041, 2004.

[FK99] Paul W Finn and Lydia E Kavraki. Computational approaches to drug design. *Algorithmica*, 25(2-3):347–371, 1999.

[FKS06] D. Ferguson, N. Kalra, and A. Stentz. Replanning with rrts. In *ICRA*, pages 1243–1248, 2006.

[FL04] E. Ferre and J.-P. Laumond. An iterative diffusion algorithm for part disassembly. In *ICRA*, pages 3149–3154, 2004.

[GFC09] J. Guitton, J.-L. Farges, and R. Chatila. Cell-rrt: Decomposing the environment for better plan. In *IROS*, pages 5776–5781, 2009.

[GSL12] Kalin Gochev, Alla Safonova, and Maxim Likhachev. Planning with adaptive dimensionality for mobile manipulation. In *ICRA*, pages 2944–2951. IEEE, 2012.

[HJRS03] D. Hsu, Tingting Jiang, J. Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *ICRA*, pages 4420–4426, 2003.

[HKLR02a] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

[HKLR02b] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *Int'l. Journal of Robotics Research*, 21(3):233–255, 2002.

[HLK06] D. Hsu, J.C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *IEEE Transactions on Robotics*, 25(7):627–643, 2006.

[HSAlCL06] D. Hsu, G. Sanchez-Ante, Ho lun Cheng, and J.-C. Latombe. Multi-level free-space dilation for sampling narrow passages in prm planning. In *ICRA*, pages 1255–1260, 2006.

[Jol86] I. Jolliffe. Principle component analysis. In *Springer-Veriag*, 1986.

[KE05] Jongwoo Kim and Joel M. Esposito. An rrt-based algorithm for testing and validating multi-robot controllers. In *In Robotics: Science and Systems*, pages 249–256, 2005.

[KF10] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *CoRR*, abs/1005.0416, 2010.

[KH06] H. Kurniawati and D. Hsu. Workspace-based connectivity oracle: An adaptive sampling strategy for prm planning. In *Int'l. Workshop on the Algorithmic Foundations of Robotics*, 2006.

[Kha86] O. Khatib. Real-time obstable avoidance for manipulators and mobile robots. *Int'l. Journal of Robotics Research*, 5(1):90–98, 1986.

[KHH$^+$09] Duksu Kim, Jae-Pil Heo, Jaehyuk Huh, John Kim, and Sung-Eui Yoon. HPCCD: Hybrid parallel continuous collision detection. *Comput. Graph. Forum (Pacific Graphics)*, 28(7), 2009.

[KJKN$^+$02] James J Kuffner Jr, Satoshi Kagami, Koichi Nishiwaki, Masayuki Inaba, and Hirochika Inoue. Dynamically-stable motion planning for humanoid robots. *Autonomous Robots*, 12(1):105–118, 2002.

[KK06] R.A. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables. In *IROS*, pages 3375–3380, 2006.

[KL00] J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *ICRA*, pages 995–1001, 2000.

[KSLO96] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[KvdP06] M. Kalisiak and M. van de Panne. Rrt-blossom: Rrt with a local flood-fill behavior. In *ICRA*, pages 1237 –1242, 2006.

[Lat99] J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *Int'l. Journal of Robotics Research*, 18:1119–1128, 1999.

[LaV06] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[LF09] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *Int'l. J. Rob. Res.*, 28(8):933–945, 2009.

[LFG$^+$05] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[LGLM99] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of North Carolina, 1999.

[Lie77] Alain Liegeois. Automatic supervisory control of the configuration and behavior of multi-body mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(12):868–871, 1977.

[LJTM94] J-P Laumond, Paul E Jacobs, Michel Taix, and Richard M Murray. A motion planner for nonholonomic mobile robots. *Robotics and Automation, IEEE Transactions on*, 10(5):577–593, 1994.

[LK01] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[LKZY12] Junghwan Lee, OSung Kwon, Liangjun Zhang, and Sungeui Yoon. Sr-rrt: Selective retraction-based rrt planner. In *ICRA*, pages 2543–2550, 2012.

[LL04] S.R. Lindemann and S.M. LaValle. Incrementally reducing dispersion by increasing voronoi bias in rrts. In *ICRA*, pages 3251–3257, 2004.

[LSL99] Florent Lamiraux, Sepanta Sekhavat, and J-P Laumond. Motion planning and control for hilare pulling a trailer. *Robotics and Automation, IEEE Transactions on*, 15(4):640–652, 1999.

[MK85] Anthony A Maciejewski and Charles A Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *Int'l. Journal of Robotics Research*, 4(3):109–117, 1985.

[MTP$^+$05] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N.M. Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, volume 17, pages 361–376, 2005.

[NAS] NASA - 3d resources. `http://www.nasa.gov/multimedia/3d_resources/`.

[NHY87] Yoshihiko Nakamura, Hideo Hanafusa, and Tsuneo Yoshikawa. Task-priority based redundancy control of robot manipulators. *Int'l. Journal of Robotics Research*, 6(2):3–15, 1987.

[ope] Open Dynamics Engine. `http://sourceforge.net/projects/opende/files/`.

[PK05] M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *IROS*, pages 3231–3237, 2005.

[PKS$^+$11] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. R. Walter. Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In *IROS*, pages 4307–4313, 2011.

[PKV10] Erion Plaku, EE Kavraki, and Moshe Y Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *Robotics, IEEE Transactions on*, 26(3):469–482, 2010.

[PZM10] Jia Pan, Liangjun Zhang, and Dinesh Manocha. Retraction-based rrt planner for articulated models. In *ICRA*, pages 2529–2536, 2010.

[Rei79] John H. Reif. Complexity of the mover's problem and generalizations. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:421–427, 1979.

[RL05] S. Redon and M.C. Lin. Practical local planning in the contact space. In *ICRA*, pages 4200–4205, 2005.

[RTLA06] Samuel Rodriguez, Xinyu Tang, Jyh-Ming Lien, and Nancy M Amato. An obstacle-based rapidly-exploring random tree. In *ICRA*, pages 895–900, 2006.

[RTPA06] S. Rodriguez, S. Thomas, R. Pearce, and N. Amato. Resampl: A region-sensitive adaptive motion planner. In *Int'l. Workshop on the Algorithmic Foundations of Robotics*, 2006.

[SB02] Cyrill Stachniss and Wolfram Burgard. An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In *IROS*, volume 1, pages 508–513. IEEE, 2002.

[SD91] Zvi Shiller and Steven Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *Robotics and Automation, IEEE Transactions on*, 7(6):785–797, 1991.

[SHJ$^+$05] Zheng Sun, D. Hsu, Tingting Jiang, H. Kurniawati, and J.H. Reif. Narrow passage sampling for probabilistic roadmap planning. *IEEE Transactions on Robotics*, 21(6):1105–1115, 2005.

[SIB03] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547, 2003.

[SK05] Luis Sentis and Oussama Khatib. Control of free-floating humanoid robots through task prioritization. In *ICRA*, pages 1718–1723, 2005.

[SK08] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2008.

[SK12] Ioan Sucan and Lydia E Kavraki. A sampling-based tree planner for systems with complex dynamics. *Robotics, IEEE Transactions on*, 28(1):116–131, 2012.

[SL05] M. Saha and J.-C. Latombe. Finding narrow passages with probabilistic roadmaps: the small step retraction method. In *IROS*, pages 622–627, 2005.

[SLN00] T. Simeon, J. P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6), 2000.

[SMK12] Ioan A. Sucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`.

[SO97] Petr Svestka and Mark H Overmars. Motion planning for carlike robots using a probabilistic learning approach. *The International Journal of Robotics Research*, 16(2):119–143, 1997.

[ST09] Alexander Shkolnik and Russ Tedrake. Path planning in 1000+ dimensions using a task-space voronoi bias. In *ICRA*, pages 2061–2067, 2009.

[ST11] A. Shkolnik and R. Tedrake. Sample-based planning with volumes in configuration space. *CoRR*, abs/1109.3145, 2011.

[SWT09] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. Reachability-guided sampling for planning under differential constraints. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2859–2865. IEEE, 2009.

[VWFS07] M Vande Weghe, Dave Ferguson, and Siddhartha S Srinivasa. Randomized path planning for redundant manipulators without inverse kinematics. In *7th IEEE-RAS Int'l. Conf. on Humanoid Robots*, pages 477–482, 2007.

[WB11a] N.A. Wedge and M.S. Branicky. An obstacle-responsive technique for the management and distribution of local rapidly-exploring random trees. In *IROS*, pages 2620 –2625, sept. 2011.

[WB11b] N.A. Wedge and M.S. Branicky. Using path-length localized rrt-like search to solve challenging planning problems. In *ICRA*, pages 3713 –3718, may 2011.

[YG07] Zhenwang Yao and Kamal Gupta. Path planning with general end-effector constraints. *Robotics and Autonomous Systems*, 55(4):316–327, 2007.

[YJSL05] A. Yershova, L. Jaillet, T. Simeon, and S.M. LaValle. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. In *ICRA*, pages 3856–3861, 2005.

[YTEA12] Hsin-Yi Yeh, Shawna Thomas, David Eppstein, and Nancy M. Amato. Uobprm: A uniformly distributed obstacle-based prm. In *IROS*, 2012.

[YYN11] Katsu Yamane, Yoshifumi Yamaguchi, and Yoshihiko Nakamura. Human motion database with a binary tree and node transition graphs. *Autonomous Robots*, 30(1):87–98, 2011.

[ZM08] L. Zhang and D. Manocha. An efficient retraction-based rrt planner. In *ICRA*, pages 3743–3750, 2008.

Summary

Sampling-based motion planning algorithm to handle a narrow passage problem

로봇 모션 플래닝은 부품 조립, 무인자동차, 컴퓨터 그래픽스 등 많은 분야에서 널리 사용되는 문제로 1970년대 이래로 활발하게 연구가 된 분야이다. 그중에서도 확률적 완전성(probabilistically complete) 특징을 갖는 샘플링 기반 알고리즘이 다양한 종류의 로봇과 제약조건에 대해 많이 사용되어 왔다. 그러나, 샘플링 기반 알고리즘은 문제 환경 내에 좁은 길이 포함되는 경우에 성능이 저하되는 문제점을 가지고 있다. 이러한 경향은 운동학적 동역학적 제약조건들이 추가되거나, 로봇 매니퓰레이터(robot manipulator) 처럼 로봇의 자유도가 높아지는 경우 더욱 악화된다. 본 학위논문에서 자유비행 강체로봇(free-flying rigid robot)의 경로를 다양한 특성을 가진 환경에 대해 효율적으로 수행하는 알고리즘과 로봇 매니퓰레이터를 위한 새로운 작업공간(task-space) 기반 알고리즘을 제안한다. 강체로봇을 위한 알고리즘에서 새롭게 제안하는 브리지 라인테스트를 이용해 환경에서 좁은길의 존재여부를 탐색하여 필요한 영역에서만 리트렉션 방법을 선택적으로 수행한다. 또한 넓은 오픈 프리 스페이스(open free-space) 영역에서의 불필요한 샘플링을 제외시키는 방법으로 비충돌 라인테스트를 제안한다. 제안된 두개의 라인테스트는 계산량이 적으며, 이를 이용해 좁은 길을 포함한 경우나 그렇지 않은 경우 모두 기존에 알려진 방법보다 더 빠른 속도로 문제를 해결할 수 있다. 로봇 매니퓰레이터를 위한 효율적인 알고리즘을 위해, 먼저 작업공간 내에서 로봇을 목적하는 지점까지 옮기기 위한 효율적인 위치의 집합을 생산적 구역(productive regions)으로 정의한다. 생산적 구역 내에 샘플이 포함되는지를 알기위해 해당 샘플 위치에서의 최대도달가능지역을 계산한다. 최대도달가능지역에 목적지가 포함되는 경우에는 이를 유망한 지역으로 정의하고 유망한 최대도달가능지역들에서 샘플링을 집중한다. 반면에 목적지가 포함되지 않는 경우에는 비유망 지역으로 이를 정의하고 우회경로 구역을 계산하여 이곳에서 샘플링을 집중한다. 우회경로 구역과 유망한 최대도달가능지역에 편향된 샘플링을 통해 더 효율적으로 경로를 계산해 낼 수 있다.

감 사 의 글

가장 먼저 지금까지 키워주시고, 항상 응원해주신 사랑하는 부모님께 감사의 마음을 전합니다.

그리고 박사과정 동안 조언을 아끼지 않으시고, 연구를 지도해주신 윤성의 교수님께 감사 드립니다. 교수님의 관심과 지도 덕분에 연구뿐만 아니라 많은 것을 배워갈 수 있는 것 같습니다. 그리고 바쁜 시간을 쪼개어 학위 논문을 지도해주신 최성희, Otfried Cheong, 류석영, 심현철 교수님께 감사 드립니다. 그리고 제가 석사과정을 시작했을 때 지도해주신 신성용 교수님께도 깊은 감사를 드립니다. 교수님의 지도편달이 제가 연구자가 되기 위한 거대한 초석이 되었습니다.

학위를 받기까지 수년간 동거동락 한 SGLAB 식구들 에게도 감사의 말을 전합니다. 연구실 1호 박사 김오빠 태준, 이런 저런 일을 너무 많이 함께해 일일이 적기도 힘든 덕수, 새신랑 보창과 Pio, 양꼬치를 좋아하는 재필, 다들 고맙고 이제 자주 볼 수 없다니 마음이 아픕니다. 그리고 우리 수많은 후배님들 돌고 돌아 같은 곳에 있는 용영과 종협, 종윤, 미소아빠 근호, Lin, 오성형, 귀염둥이 효섭, 마른 장작 영운, 명환, 시간이 더 흐르기 전에 계속 자주 봤으면 좋겠네요. 그리고 앞으로 연구실을 끌어갈 친구들, CTO 정인, 기둥 동혁, 여자 3총사 가연, 수민, Mingyang, 명배, 창민, 그리고 핫한 신입생들 웅직, 현철, 용선, 태영, 다들 좋은 연구하고 졸업하길 바랍니다. 그리고 박은재, 남효정 선생님, 그리고 영현에게도 감사의 말을 전합니다. 또한 함께 시간을 공유했던 TC group 식구들 진호형, 봉철형, 태수형, 영상형, 나협형, 내진형, 지현누나, 윤형형, 춘석형, 재환형, 신행형, 용준, 상욱형, 거엽, 민혁, 동훈, 다성, 일우, 상호 에게도 감사의 말을 전합니다. 그 외 일일이 열거하지 못한 수많은 동기 및 선후배들에게 감사의 말을 전합니다.

길었던 대학원 생활 동안 기쁨과 슬픔을 함께 해주었던 수많은 분들께도 감사의 말을 전합니다. 공부한다며 대전으로 훌렁 떠나버려 가끔씩 볼 수 밖에 없었던 서울 친구들, 쓸쓸한 대전 생활에 활력소가 되었던 스윙피버 사람들, 그리고 귀염둥이 정순이에게 감사의 말을 전합니다.

이 력 서

이　　　름 :	이 정 환
생 년 월 일 :	1982년 10월 5일
E-mail 주 소 :	goolbee@kaist.ac.kr

학　　력

1998. 3. – 2001. 2.	경기고등학교
2001. 3. – 2006. 2.	연세대학교 컴퓨터과학과 (B.S.)
2006. 3. – 2014. 8.	한국과학기술원 전산학과 (Integrated M.S. and Ph.D.)

연 구 업 적

1. **Junghwan Lee**, OSung Kwon, Liangjun Zhang, and Sung-eui Yoon, *SR-RRT: Selective Retraction-based RRT Planner*, IEEE International Conference on Robotics and Automations (ICRA), 2012
2. Duksu Kim, Jinkyu Lee, **Junghwan Lee**, InSik Shin, John Kim, Sung-eui Yoon *Scheduling in Heterogeneous Computing Environments for Proximity Queries*, IEEE Transactions on Visualization and Computer Graphics (TVCG), 2013
3. Tieu Lin Loi, Jae-Pil Heo, **Junghwan Lee** and Sung-Eui Yoon *VLSH: Voronoi-based Locality Sensitive Hashing*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013
4. **Junghwan Lee** and Sung-Eui Yoon *PROT: Productive Regions-Oriented Task space path planning for hyper-redundant manipulators*, IEEE International Conference on Robotics and Automations (ICRA), 2014
5. Donghyuk Kim, **Junghwan Lee** and Sung-Eui Yoon *Cloud RRT*: Sampling Cloud based RRT** , IEEE International Conference on Robotics and Automations (ICRA), 2014
6. **Junghwan Lee**, OSung Kwon, Liangjun Zhang, and Sung-eui Yoon *Selective retraction-based RRT planner for various environments*, IEEE Transactions on Robotics, 2014