Short Paper

A Selective Retraction-Based RRT Planner for Various Environments

Junghwan Lee, OSung Kwon, Liangjun Zhang, and Sung-Eui Yoon

***Abstract*—We present a novel randomized path planner for rigid robots to efficiently handle various environments that have different characteristics. We first present a *bridge line test* that can identify narrow passage regions and then selectively performs an optimization-based retraction only at those regions. We also propose a *noncolliding line test*, which is 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. We have tested our method with different benchmarks that have varying amounts of narrow passages. Our method achieves up to several times improvements over prior RRT-based planners and consistently shows the best performance across all the tested benchmarks.**

***Index Terms*—Motion and path planning, sampling-based motion planning, various environments.**

I. Introduction

Robot motion planning involves computing collision-free paths for a robot and has many applications, such as part disassembly simulation, tolerance verification, protein folding, and computer graphics [1], [2]. The motion planning has been actively studied since the 1970s to develop a complete solution.

Sampling-based motion planning algorithms have been designed and successfully used to compute a probabilistic complete solution for a variety of environments. Two of the most successful algorithms include the Probabilistic Roadmap Method (PRM) [3] and Rapidly-exploring Random Tree (RRT) [4]. At a high level, as we randomly generate more samples, 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 many practical motion planning applications, a robot often needs to pass through narrow passages and the performance of sampling-based algorithms can degrade significantly. Many approaches have been proposed in different directions such as utilizing the workspace geometric information [5], filtering (or adaptive sampling) techniques toward more important regions [6]–[8], retracting samples, etc.

Recently, retraction-based techniques [9]–[13] have been actively studied, since they can pose samples near the boundary of C-Obstacles, efficiently leading to explore important regions including narrow passages. However, these techniques have the computation overhead of retracting sampling and, thus, can run even slower than a basic RRT when the free space does not have such difficult regions [9].

Main results: In this paper, we propose a novel retraction-based planner for rigid robots known as *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. In addition, in order to generate more random samples near narrow passages, we present a *noncolliding line test* that detects wide-open free spaces and cull samples near such spaces (see Section IV).

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 (see Section V). 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 two times on average) over a basic RRT [4], a dynamic-domain RRT (DD-RRT) [8], and an optimization-based retraction method [9], 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.

A preliminary version of this paper has appeared in ICRA 2012 [14]. In this paper, we have additionally compared proposed algorithms with an alternative sampling-based technique, the DD-RRT. We have also performed in-depth analysis on both the results with several performance metrics and the relative contributions of each algorithmic component. For better exploitation, we have also added implementation details of the algorithm.

II. Related Work

In this section, we discuss prior work on sampling-based motion planning that has been especially designed to efficiently handle narrow passages.

A. Sampling-Based Motion Planning

The sampling-based algorithms have been successfully used to solve various motion planning problems in practice. Among them, the Probabilistic Roadmap Method (PRM) [3] and Rapidly-exploring Random Tree (RRT) [4] are the most widely used approaches [15]. These techniques have been extensively studied, and an excellent survey is available [16].

Our method is built upon the RRT methods, since they have been extended to solve a wide variety of practical single-query problems

Manuscript received November 22, 2013; accepted February 28, 2014. This paper was recommended for publication by Associate Editor M. Vendittelli and Editor D. Fox upon evaluation of the reviewers' comments. This work was supported in part by NRF-2013R1A1A2058052, DAPA/ADD (UD110006MD), MEST/NRF (2013-067321), and the IT R&D program of MOTIE/KEIT [10044970].

J. Lee, O. Kwon, and S.-E. Yoon are with the Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea (e-mail: goolbee@gmail.com; miruce78@gmail.com; sungeui@gmail.com).

L. Zhang is with Samsung Research America, San Jose, CA 95134 USA (e-mail: zlj@cs.unc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2014.2309836

in robotics and other related domains [17]–[20]. RRT methods have been improved in many different directions by considering workspace information [5], [21], biasing sampling strategies [8], [22], and decomposing environments and focusing sampling on critical paths [23], taking into account the optimality of solutions [24], etc.

B. Narrow Passages

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 [15]. Many strategies have been proposed to improve the performance on these difficult problems [16].

Many PRM or RRT variations have tried 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 [5], [25], dynamic sampling distributions [26], simultaneously mapping roadmaps at C-free and C-Obstacle spaces [27], utilizing local free space information [28], filtering techniques toward important regions including narrow passages [6]–[8], [29]–[33], and retraction-based approaches [9]–[13].

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.* [30] proposed the Gaussian sampling strategy that distributes samples near the boundary of the free space. The Bridge test proposed by Hsu *et al.* [7] uses three sampled configurations to boost the sampling density inside narrow passages. Yershova *et al.* [8] proposed a DD-RRT that adapts its sampling domain to bias toward the visibility domain by obstacles. Obstacle-based PRM proposed by Amato *et al.* [29] generates a ray from invalid samples to find a surface configuration, and UOBPRM [32] improves the performance by generating surface configurations uniformly. Shkolnik and Tedrake [33] proposed the ball tree algorithm that approximates the reachable free space in a similar way to our noncolliding line test.

Retraction-based techniques: The main idea of the retraction-based approach is to retract initial samples normally generated from uniform sampling on the C-space toward more useful regions such as the boundary of the 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 [9], [10], [12], dilate the free space [11], [13], or use the approximation of medial axes or boundary of the motion for the robot [34]. 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.

At a high level, our approach integrates these two different approaches, i.e., filtering and retraction-based techniques, in order to robustly handle various environments that do or do not have narrow passages. The new integrated planner, i.e., SR-RRT, utilizes the retraction technique to bias the uniform sampling distribution inherited from the standard RRT and reduces unpromising retraction operations by using a relatively cheap filtering method in order to improve the overall performance of the planner.

Hsu *et al.* [35] also considered to use different sampling methods and adaptively used them for a higher performance. Unlike this approach, this paper explores another direction of integrating different techniques; our method uses a filtering method as a *culling technique* to decide whether it is desirable to perform an expensive retraction method or not. Our work exploits an optimization-based retraction technique [9]

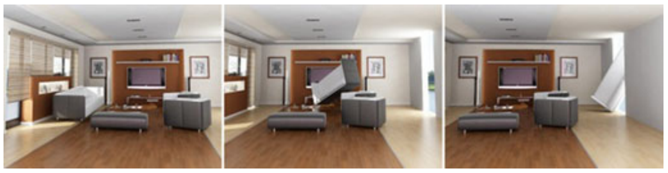

Fig. 1. *Living room benchmark:* This figure shows three image shots of getting a sofa out through a door, which creates narrow passages in 6-degrees-of-freedom (DoFs) configuration space. Our method achieves ten and two times improvement over a basic RRT and an optimization-based retraction method, respectively.

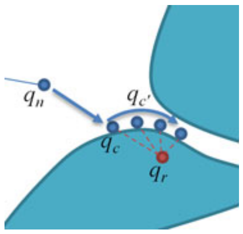

Fig. 2. *Sampling generation of an optimization-based retraction:* RRRT incrementally retracts a randomly generated in-colliding sample q_r to a more desirable place in order to generate more samples in the narrow passages. This figure shows that the original random sample q_r is in-colliding, and it is then incrementally retracted from q_c to $q_{c'}$ and so on.

as a retraction method, which has performance problems in relatively easy environments (see Section III-C). We apply PCA to our filtering methods in a similar manner as PCA-RRT [28] for increasing the accuracy of our filtering methods (see Section IV-C), while the PCA-RRT used the PCA to control random sampling.

III. Background

In this section, we give a brief review on a basic RRT and its variant based on an optimization-based retraction RRRT [9], which is designed for efficiently handling narrow passages. We then discuss their issues that arise when we apply them to various environments which have or do not have narrow passages.

A. Review of an Rapidly-Exploring Random Tree

A basic RRT algorithm starts with a single or multiple random trees [4]. The basic RRT randomly generates a sample q_r in the configuration space and finds its nearest neighbor q_n among the 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 a C-Obstacle is computed along a straight line from q_r to q_n [15]. q_c is then added to the tree and is connected with q_n (see Fig. 2).

It has been known that the basic RRT explores the free space with a bias related to the Voronoi diagram [4]. Specifically, 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, 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.

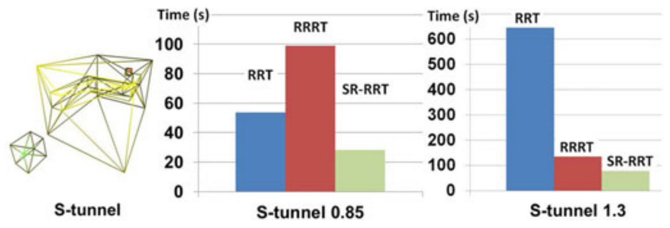

Fig. 3. *S-tunnel Benchmark:* The leftmost figure shows the S-tunnel benchmark. The right two figures show the average performance of two variations of the S-tunnel benchmark. 0.85 and 1.3 indicate the scaling factor of the cubic robot. S-tunnel 0.85 includes no narrow passages, while S-tunnel 1.3 includes them.

B. Review of an Optimization-Based Retraction Technique

In order to address the problem of the basic RRT, an optimization-based retraction technique RRRT [9] was proposed by Zhang and Manocha. Its pseudocode is shown in Algorithm 1. 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. 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. 2 shows that $q_{c'}$ is a newly retracted from in-contact configuration from q_c.

Algorithm 1 RRRT Planner

```
$T$.Init($q_{init}$)
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
```

OptimizedRetraction(q_c) used in the pseudocode of Algorithm 1 is performed as follows: First, 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 about four times higher performance than the basic RRT in the S-tunnel benchmark 1.3 (see Fig. 3), which has narrow passages.

C. 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, additional sampling, etc. These operations have relatively higher computational overheads, compared with other operations of the basic RRT method [9]. In addition, 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 (see Fig. 3) that does not have narrow passages.

IV. Selective Retraction-Based Rapidly-Exploring Random Tree

Our technique is based on the optimization-based retraction-based RRT (RRRT) method [9]. 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 [7] 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 *noncolliding line test*, a dual operator to the bridge line test, to generate more samples near narrow passages.

A. Selective Retraction-Based Extension

We first explain our extension method of SR-RRT, whose pseudocode is at Algorithm 2. Once a random sample q_r and its nearest neighbor node q_n are computed, as mentioned in Section III-A, then we perform our extension method *SRExtend*(q_n, q_r), shown in Algorithm 2. The first step of our extension method is exactly the same to the extension method of the basic RRT explained in Section III-A. Note that at this step, we may create an in-contact configuration q_c.

Algorithm 2 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*(q_c) in the pseudocode of our extension method (see Algorithm 2) and explained in Section III-B.

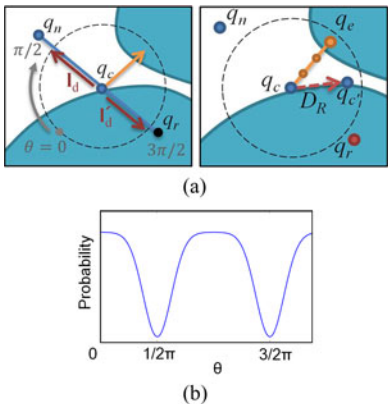

Fig. 4. (a) Two figures show the procedure of generating and performing a bridge line test. (b) Probability distribution function (PDF) for the line direction parameterized with θ. (a) Procedure of the bridge line test. (b) PDF for the line direction.

B. 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. 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.

1) Generating a Bridge Line: In particular, we use a line generating function as a probability distribution function (PDF) $p_l(\cdot)$, which is 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. 4(b) shows the 2-D 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. 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 nonzero probability to detect narrow passages with infinitely small width.

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 is 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, thus performing 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 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.

2) Retesting: 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 1-D 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 retesting is defined as *Retest*(q_n) shown in Algorithm 3.

Algorithm 3 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
```

Note that it is quite reasonable to retest 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 existing narrow passages around the node. Moreover, by allowing multiple retesting, the bridge line test can correctly identify the existence of narrow passages in a probabilistic manner with many random samples.

C. Accuracy Improvement of Bridge Line Test by Principal Component Analysis

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 1-D 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 [28], we perform the PCA [36] 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. 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 [36]. 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 [28]. When we need to generate a random line from q_c, we first collect

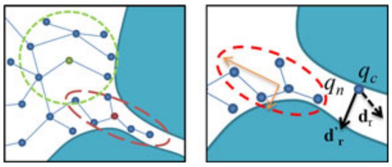

Fig. 5. (Left) Shapes of local free spaces (green and red ones) computed from two nodes. The red one located in a narrow passage has an elongated shape, while the green one located in the wide-open free space is uniformly distributed in the space. (Right) Transformed line direction $\vec{d'_r}$ from 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.

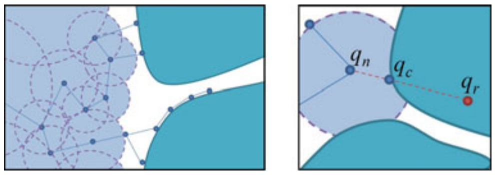

Fig. 6. (Left) Free hyperspheres that approximate wide-open free spaces. Dotted circles represent to the hypersphere with the center of node and radius of it. (Right) Example that a node q_c is generated from a random sample q_r, generated outside of hypersphere (dotted circle) by a basic RRT expansion.

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 $\vec{d_r}$, which is computed in the line generating PDF $p_l(\cdot)$ in Section IV-B, into a new line direction $\vec{d'_r}$ based on the following equation:

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

where U_i is ith 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.

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 Section IV-B. 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 $\vec{d'_r}$.

D. Noncolliding 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 have no 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. In addition, 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. 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 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 noncolliding 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 noncolliding 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.

E. Overall Algorithm

A pseudocode of the overall algorithm of our SR-RRT planner is shown in Algorithm 4. 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 (see Algorithm 2) after performing the re-testing process (see Algorithm 3). 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 4 SR-RRT Planner

```
$T$.Init($q_{init}$)
**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
```

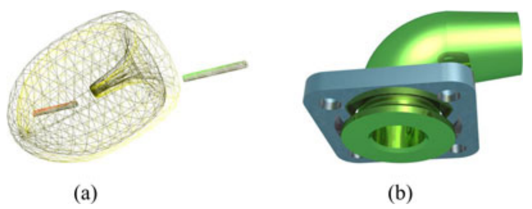

Fig. 7. Two different benchmarks that have narrow passages. Note that a robot of a bug-trap benchmark is initially located outside of a trap. (a) Bug-trap. (b) Flange.

V. Experimental Results

We have implemented SR-RRT for 3-D rigid body robots on an Intel i7 desktop machine that has 3.33 GHz CPU. Our method is built upon a basic RRT integrated with an efficient optimized-based retraction method RRRT [9]. 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.

A. 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 (see Fig. 3) has an 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 [see Fig. 7(a)], flange [Fig. 7(b)], wiper 1.0 [see Fig. 8], pipe (see Fig. 9), and living room (see Fig. 1), 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 I.

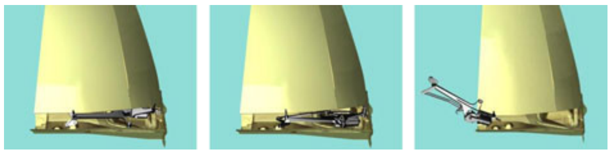

Fig. 8. Three image shots that show animation sequences of getting a wiper out of the windscreen model. Since there are not much rooms between two models, this benchmark has narrow passages.

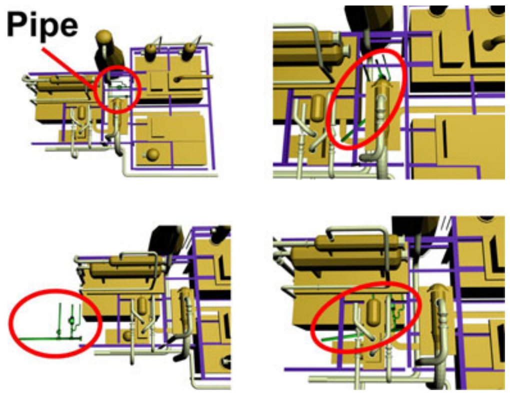

Fig. 9. Four image shots are shown for moving the pipe out of an industrial environment (clockwise starting from the top left).

B. 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 Section III-A. 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, i.e., SR-RRT, with each of our benchmarks 100 times and report the average running time in Table I 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 3 min. In the pipe and living room benchmarks, our method spends over 1 min 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 toward 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

TABLE I
PERFORMANCE OF OUR METHOD, SR-RRT (*SR-RRT*), THE BASIC RRT (*RRT*), THE DYNAMIC-DOMAIN RRT [8] (*DD-RRT*), AND THE OPTIMIZATION-BASED RETRACTION RRT (*RRRT*) [9] FOR EACH BENCHMARK MODEL WITH ITS REPRESENTATIVE IMAGE, AND MODEL COMPLEXITY, AND DIFFICULTY LEVEL

Model	# Tri	Difficulty	RRT	DD-RRT	RRRT	SR-RRT	Imp. RRT	Imp. DD-RRT	Imp. RRRT	Rep. Image
S-tunnel 0.85	64	Easy	53.68 s	58.78 s	98.83 s	28.14 s	1.91	2.09	3.51	Fig. 3
S-tunnel 1.0	64	Moderate	96.21 s	129.92 s	141.22 s	65.57 s	1.47	1.98	2.15	See above
S-tunnel 1.15	64	Hard	406 s	405.93 s	129.5 s	52.08 s	7.80	7.79	2.49	See above
S-tunnel 1.3	64	Hard	646 s	849.92 s	134.72 s	78.9 s	8.19	10.77	1.71	See above
Bug-trap	2.7k	Hard	145 s	35.45 s	39.72 s	22.17 s	6.54	1.60	1.79	Fig. 7-(a)
Flange	6.3k	Hard	589.24 s	875.25 s	33.12 s	27.99 s	21.05	31.27	1.18	Fig. 7-(b)
Wiper 0.9	26.7k	Moderate	107.57 s	63.99 s	107.75 s	48.05 s	2.24	1.33	2.24	Fig. 8
Wiper 1.0	26.7k	Hard	283.56 s	143.27 s	141.5 s	82.91 s	3.42	1.73	1.71	See above
Pipe	48.4k	Hard	639.99 s	1001.97 s	225.79 s	166.08 s	3.85	6.03	1.36	Fig. 9
Livingroom	137k	Hard	776.54 s	242.94 s	140.94 s	77.2 s	10.06	3.15	1.83	Fig. 1

The table also shows improvements (Imp.) of our method over *RRT, DD-RRT*, and *RRRT*. See Section V for the detailed information.

TABLE II
NUMBER OF ITERATIONS AND THE CONTRIBUTION OF EACH COMPONENT ON SR-RRT

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

Incremental effects by enabling each component on each tested benchmark BL-test, NC-test, and PCA represent to the bridge line test (see Section IV-B) , noncolliding line test (see Section IV-D), and PCA-transformed line direction selection (see Section IV-C), respectively.

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 DD-RRT [8] 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. The DD-RRT performs well in general compared with RRT in many environments, but its performance rather depends on the size of the sampling domain specified for an environment. For example, when the space of C-obstacles is relatively large compared with the sampling domain, DD-RRT shows worse performance than RRT. While the performance of the DD-RRT is dependent on environments, our planner shows better performance with all the benchmarks with or without narrow passages. Our planner shows 6.8 times performance improvement overall compared with DD-RRT (See Table I).

Table II 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 s) shows four times slower performance than one iteration of RRT (0.0168 s). Our planner, i.e., SR-RRT, reduces the number of retraction iterations, which takes a relative amount of 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 s) is similar to RRRT. As a result, our method shows higher performance than RRT and RRRT.

VI. DISCUSSIONS

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

A. *Contributions of Each Component*

We measure how much improvement we make with each component of our contributions. By enabling bridge line tests (see Section IV-B) only, we observe 74.6% improvement over RRRT. We achieve 5.48% further improvement on average by additionally enabling noncolliding tests (see Section IV-D). In addition, by adopting the PCA-transformed line direction (see Section IV-C), we observe additional 5.04% improvement on average. Table II 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 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 noncolliding 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 the space is relatively wide-open free space. Our tested benchmarks are mostly hard-typed ones that include narrow passages and fewer widely open free spaces.

B. *Breakdown of Running Time*

Table III 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 noncolliding line tests are quite high (e.g., 78% to 97%) across all the benchmarks. On the

TABLE III
BREAKDOWN OF THE RUNNING TIME OF THE SR-RRT

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%

Cull % refers to the culling ratios of two types of line-tests.

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.

C. Statistical Error of Two Line Tests

Proposed two line tests, i.e., bridge and noncolliding 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, i.e., a false positive, is zero, because a bridge line is never found when there is no narrow passage (see Section IV-B1). On the other hand, we can have a type II error, i.e., 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 noncolliding line test (see Section IV-D). When a type I error occurs, it rejects a sample inside of a nonfree 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 noncolliding 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 (see Section III-A). The right-hand side of Fig. 6 shows an example.

D. Implementation of Principal Component Analysis Operations

Table IV shows portions of time spent for performing PCA compared with the overall running time and the bridge line test. We implemented an incremental PCA [37] in order to reduce the running time of PCA operation by eliminating recomputation of the diagonalization of the covariance matrix [28]. The PCA operation still takes a large portion of the bridge line test, but its portion of the overall running time is relatively low (6% on average).

TABLE IV
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

A parameter k, which is 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.

E. Implementation Issues

When we compute a probability distribution function $p_d(\cdot)$ to sample an end point of a bridge line test (see Section IV-B1), we use the Gaussian function with the mean used for an optimization-based retraction operation D_R. The main reason for 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 as employed in the local planner.

F. Analysis With Varying Scaling Factors

We have tested S-tunnel models with varying scaling factors for the cubic-shaped robot (see Table I). 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 the 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.

G. 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. In addition, 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. In addition, 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.

VII. Conclusion and Future Works

We have presented a novel retraction-based planner, i.e., SR-RRT, which selectively performs the retraction operations only near narrow passages. To perform such adaptive retraction operations, we proposed a bridge line test that can efficiently identify whether a region contains narrow passages or not. We perform PCA with local free spaces and generate lines that can cross potentially existing narrow passages with a high probability. In addition, in order to generate samples near such narrow passages, we presented a noncolliding line test that can identify whether a region is a wide-open free space or not. These line tests have minor computational overheads and can work for high dimensional configuration spaces. Moreover, our method has been demonstrated to show up to 21 times, 31 times, and 3.5 times (6.7 times, 6.8 times, and two times on average) performance improvement compared with DD-RRT, a basic RRT, and an optimization based RRT method, respectively. More importantly, 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.

There are many avenues for future research directions. We would like to design more accurate, yet efficient filtering methods. In addition, we would like to identify collision-free paths in a multiresolution approach [23], [25] in order to more efficiently find such paths. It will be very challenging to design a multiresolution technique for environments that contain narrow passages shown in this paper. Our planner requires some parameters, including ones inherited from the retraction operation. Eliminating the effort to tune the algorithm can be beneficial. In addition, adopting better sampling techniques [38] or nearest neighbor search methods [39] can further accelerate the performance of our approach. Finally, the optimization-based retraction method has been extended to articulated robots [40]. We would also like to extend and test our method to such cases.

References

[1] P. W. Finn and L. E. Kavraki, “Computational approaches to drug design,” *Algorithmica*, vol. 25, no. 2–3, pp. 347–371, 1999.

[2] J. Latombe, “Motion planning: A journey of robots, molecules, digital actors, and other artifacts,” *Int. J. Robot. Res.*, vol. 18, pp. 1119–1128, 1999.

[3] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[4] J. Kuffner and S. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000, pp. 995–1001.

[5] H. Kurniawati and D. Hsu, “Workspace-based connectivity oracle: An adaptive sampling strategy for PRM planning,” in *Proc. Int. Worksh. Algorithm. Found. Robot.*, 2006.

[6] T. Simeon, J. P. Laumond, and C. Nissoux, “Visibility based probabilistic roadmaps for motion planning,” *Adv. Robot. J.*, vol. 14, no. 6, pp. 477–493, 2000.

[7] D. Hsu, T. Jiang, J. Reif, and Z. Sun, “The bridge test for sampling narrow passages with probabilistic roadmap planners,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2003, pp. 4420–4426.

[8] A. Yershova, L. Jaillet, T. Simeon, and S. LaValle, “Dynamic-domain RRTS: Efficient exploration by controlling the sampling domain,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2005, pp. 3856–3861.

[9] L. Zhang and D. Manocha, “An efficient retraction-based rrt planner,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 3743–3750.

[10] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato, “An obstacle-based rapidly-exploring random tree,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2006, pp. 895–900.

[11] D. Hsu, G. Sanchez-Ante, H. lun Cheng, and J.-C. Latombe, “Multi-level free-space dilation for sampling narrow passages in prm planning,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2006, pp. 1255–1260.

[12] S. Redon and M. Lin, “Practical local planning in the contact space,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2005, pp. 4200–4205.

[13] M. Saha and J.-C. Latombe, “Finding narrow passages with probabilistic roadmaps: The small step retraction method,” in *Proc. Int. Conf. Intell. Robots Syst.*, 2005, pp. 622–627.

[14] J. Lee, O. Kwon, L. Zhang, and S. Yoon, “SR-RRT: Selective retraction-based RRT planner,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 2543–2550.

[15] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[16] D. Hsu, J. Latombe, and H. Kurniawati, “On the probabilistic foundations of probabilistic roadmap planning,” *Int. J. Robot. Res.*, vol. 25, no. 7, pp. 627–643, 2006.

[17] J. Bruce and M. Veloso, “Real-time randomized path planning for robot navigation,” in *Proc. Int. Conf. Intell. Robots Syst.*, 2002, pp. 2383–2388.

[18] D. Ferguson, N. Kalra, and A. Stentz, “Replanning with RRTs,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2006, pp. 1243–1248.

[19] M. Branicky, M. Curtiss, J. Levine, and S. Morgan, “Sampling-based planning, control and verification of hybrid systems,” *Proc. Inst. Elect., Control Theory Appl.*, vol. 153, pp. 575–590, 2006.

[20] J. Cortes, L. Jaillet, and T. Simeon, “Molecular disassembly with RRT-like algorithms,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2007, pp. 3301–3306.

[21] B. Burns and O. Brock, “Single-query motion planning with utility-guided random trees,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2007, pp. 3307–3312.

[22] S. Lindemann and S. LaValle, “Incrementally reducing dispersion by increasing voronoi bias in RRTS,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, pp. 3251–3257.

[23] J. Guitton, J.-L. Farges, and R. Chatila, “Cell-RRT: Decomposing the environment for better plan,” in *Proc. Int. Conf. Intell. Robots Syst.*, 2009, pp. 5776–5781.

[24] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[25] S. Rodriguez, S. Thomas, R. Pearce, and N. Amato, “RESAMPL: A region-sensitive adaptive motion planner,” in *Proc. Int. Workshop Algorithm. Found. Robot.*, 2006, pp. 4037–4044.

[26] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. Amato, “A machine learning approach for feature-sensitive motion planning,” in *Algorithm. Found. Robot. VI*. vol. 17, Berlin, Germany: Springer, 2005, pp. 361–376.

[27] J. Denny and N. M. Amato, “Toggle PRM: Simultaneous mapping of C-free and C-obstacle—A study in 2D,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 2632–2639.

[28] S. Dalibard and J. Laumond, “Linear dimensionality reduction in random motion planning,” *Int. J. Robot. Res.*, vol. 30, no. 12, pp. 1461–1476, 2011.

[29] N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo, “OBPRM: An obstacle-based PRM for 3D workspaces,” in *Proc. Int. Worksh. Algorithm. Found. Robot.*, 1998, pp. 197–204.

[30] V. Boor, M. Overmars, and A. van der Stappen, “The Gaussian sampling strategy for probabilistic roadmap planners,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 1999, pp. 1018–1023.

[31] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. Reif, “Narrow passage sampling for probabilistic roadmap planning,” *IEEE Trans. Robot.*, vol. 21, no. 6, pp. 1105–1115, Dec. 2005.

[32] H.-Y. Yeh, S. Thomas, D. Eppstein, and N. M. Amato, “UOBPRM: A uniformly distributed obstacle-based PRM,” in *Proc. Int. Conf. Intell. Robots Syst.*, 2012, pp. 2653–2662.

[33] A. Shkolnik and R. Tedrake, “Sample-based planning with volumes in configuration space,” *CoRR*, 2011. Available http://arvix.org/abs/1109.3145.

[34] E. Ferre and J.-P. Laumond, “An iterative diffusion algorithm for part disassembly,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, pp. 3149–3154.

[35] D. Hsu, G. Sanchez-Ante, and Z. Sun, “Hybrid PRM sampling with a cost-sensitive adaptive strategy,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2005, pp. 3874–3880.

[36] I. Jolliffe, *Principle Component Analysis*. New York, NY, USA: Springer-Veriag, 1986.

[37] D. Erdogmus, Y. N. Rao, H. Peddaneni, A. Hegde, and J. C. Principe, "Recursive principal components analysis using eigenvector matrix perturbation," *EURASIP J. Appl. Signal Process.*, vol. 2004, pp. 2034–2041, 2004.

[38] D. Kim, J. Lee, and S. Yoon, "Cloud RRT*: Sampling cloud based RRT*," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014.

[39] T. L. Loi, J.-P. Heo, J. Lee, and S.-E. Yoon, "VLSH: Voronoi-based locality sensitive hashing," in *Proc. Int. Conf. Intell. Robots Syst.*, 2013, pp. 2543–2550.

[40] J. Pan, L. Zhang, and D. Manocha, "Retraction-based RRT planner for articulated models," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 2529–2536.