



Chapter 7 - Sampling-Based Algorithms

Principles of Robot Motion: Theory, Algorithms, and Implementation

by Howie Choset et al.

The MIT Press © 2005

◀ Previous

Next ▶

Chapter 7: Sampling-Based Algorithms

Overview

Different Planners described in [chapter 5](#) build roadmaps in the free (or semi-free) configuration space. Each of these methods relies on an explicit representation of the geometry of Q_{free} . Because of this, as the dimension of the configuration space grows, these planners become impractical. [Figure 7.1](#) shows a path-planning problem that cannot be solved in a reasonable amount of time with the methods presented in [chapter 5](#), but can be solved with the *sampling-based* methods described in this chapter. Sampling-based methods employ a variety of strategies for generating samples (collision-free configurations of the robot) and for connecting the samples with paths to obtain solutions to path-planning problems.

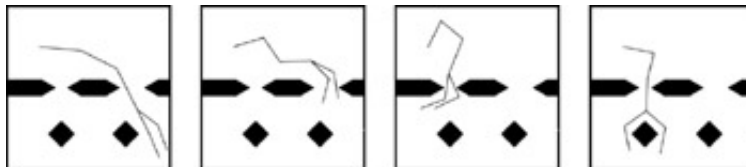


Figure 7.1: Snapshots along a path of a planar manipulator with ten degrees of freedom. The manipulator has a fixed base and its first three links have prismatic joints—they can extend to one and a half times their original length. (From Kavraki [221].)

[Figures 7.2\(a\)](#) and [\(b\)](#) show two typical examples from industrial automation that sampling-based planners can routinely solve. Sampling-based planners can also be used to address problems that extend beyond classic path planning. [Figure 7.2\(c\)](#) shows a CAD (computer-aided design) model of an aircraft engine. A planner can be used to determine if a part can be removed from that engine. Such information is extremely important for the correct design of the engine, as certain parts need to be removable for maintainability purposes. In this case, the planner considers the part to be separated as a robot that can move freely in space. [Figure 7.2\(d\)](#) involves an example from computer animation where a planner is used to plan the motion of the human figure. [Figures 7.2\(e\)](#) and [\(f\)](#) provide examples that involve planning with kinematic and dynamic constraints, while [figure 7.2\(g\)](#) displays the folding of a small peptide molecule. This chapter discusses the basics of sampling-based path planning.

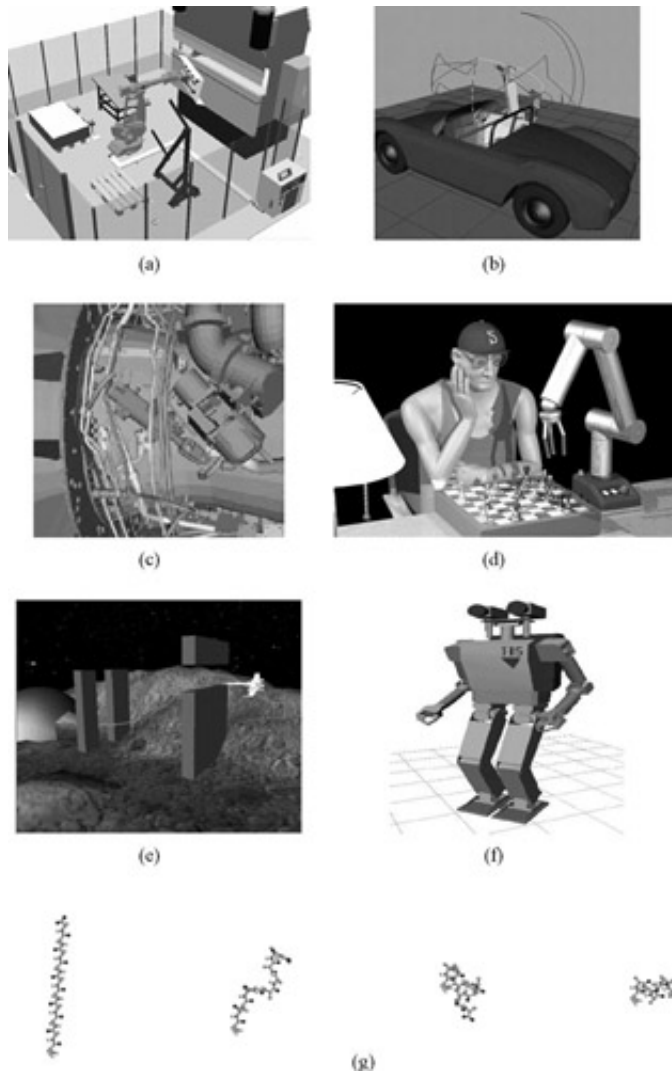


Figure 7.2: Path-planning problems. (a) Industrial manipulation. (b) Welding. (c) Planning removal paths for a part (the "robot") located at the center of the figure. (d) Computer animation. (e) Planning aircraft motion. (f) Humanoid robot. (g) Folding of a small peptide molecule. ((a) From Bohlin and Kavraki [54]; (b) from Hsu and Latombe [196]; (c) courtesy of Latombe; (d) from Koga, Kondo, Kuffner and Latombe [241]; (e) from Kuffner and LaValle [272]; (f) from Kuffner [248]; (g) from Amato [21].)

The Development of Sampling-Based Planners

Sampling-based planners were developed at a time when several complexity results on the path-planning problem were known. The generalized mover's problem, in which the robot consists of a collection of polyhedra freely linked together at various vertices, was proven PSPACE-hard by Reif [361]. Additional study on exact path-planning techniques for the generalized mover's problem led Schwartz and Sharir to an algorithm that was doubly exponential in the degrees of freedom of the robot [373]. This algorithm is based on a cylindrical algebraic decomposition of semi-algebraic descriptions of the

configuration space [117]. Recent work in real algebraic geometry renders the algorithm singly exponential [42]. Canny's algorithm [90], which builds a roadmap in the configuration space of the robot, is also singly exponential in the degrees of freedom of the robot. Furthermore, Canny's work showed that the generalized mover's problem was PSPACE-complete [90,95]. The implementation of the above general algorithms is very difficult and not practical for the planning problems shown in [figure 7.2](#).

The complexity of path-planning algorithms for the generalized mover's problem fueled several thrusts in path-planning research. These included the search for subclasses of the problem for which complete polynomial-time algorithms exist (e.g., [183, 374]), the development of methods that approximated the free configuration space (e.g., [67,68,132,297]), heuristic planners (e.g., [174]), potential-field methods (e.g., [38,40]), and the early sampling-based planners (e.g., [40,47,101,165, 220,231,244]).

The Probabilistic RoadMap planner (PRM) [231] demonstrated the tremendous potential of sampling-based methods. PRM fully exploits the fact that it is cheap to check if a single robot configuration is in Q_{free} or not. PRM creates a roadmap in Q_{free} . It uses rather coarse sampling to obtain the nodes of the roadmap and very fine sampling to obtain the roadmap edges, which are free paths between node configurations. After the roadmap has been generated, planning queries can be answered by connecting the user-defined initial and goal configurations to the roadmap, and by using the roadmap as in [chapter 5](#) to solve the path-planning problem at hand. Initially, node sampling in PRM was done using a uniform random distribution. This planner is called basic PRM. It was observed that random sampling worked very well for a wide variety of problems [221,231,345] and ensured the probabilistic completeness of the planner [221,229]. However, it was also observed [221] that random sampling is only a baseline sampling for PRM and many other sampling schemes are useful and are bound to be efficient for many planning problems as the analysis of the planner revealed. Today, these sampling schemes range from importance sampling in areas that during the course of calculations are found difficult to explore, to deterministic sampling such as quasirandom sampling and sampling on a grid. This chapter will describe the basic PRM algorithm, several popular node-sampling strategies, as well as their advantages and disadvantages, and popular node-connection strategies.

PRM was conceived as a multiple-query planner. When PRM is used to answer a single query, some modifications are made: the initial and goal configurations are added to the roadmap nodes and the construction of the roadmap is done incrementally and is stopped when the query at hand can be answered. However, PRM may not be the fastest planner to use for single queries. The second part of this chapter describes sampling-based planners that are particularly effective for single-query planning, including the Expansive-Spaces Tree planner (EST) [192, 196] and the Rapidly-exploring Random Tree planner (RRT) [249, 270]. These planners exhibit excellent experimental performance and will be discussed in detail.

Combination of the above methods is also possible and desirable in many cases. The Sampling-Based Roadmap of Trees (SRT) planner [14, 43] constructs a PRM-style roadmap of single-query-planner trees. It has been observed that for very difficult path planning problems, single-query planners need to construct large trees in order to find a solution. In some cases, the cost of constructing a large tree may be higher than the cost

of constructing a roadmap of $\mathcal{Q}_{\text{free}}$ with SRT. This illustrates the distinction between multiple-query and single-query planning, and its importance. The SRT planner will be discussed in detail in this chapter.

Despite their simplicity, which is exemplified in the basic PRM planner, sampling-based planners are capable of dealing with robots with many degrees of freedom and with many different constraints. Sampling-based planners can take into account kinematic and dynamic constraints (e.g., [195, 271]), closed-loop kinematics (e.g., [121, 184, 268]), stability constraints (e.g., [64, 247, 248]), reconfigurable robots (e.g., [98, 139, 149]), energy constraints (e.g., [251, 255]), contact constraints (e.g., [210]), visibility constraints (e.g., [123]) and others. Clearly some planners are better at dealing with specific types of constraints than others. For example, as discussed in [section 7.5.1](#), EST and RRT planners are particularly useful for problems that involve kinematic and dynamic constraints. Kinodynamic problems are described in [chapters 10, 11, and 12](#).

PRM, EST, RRT, SRT, and their variants have changed the way path planning is performed for high-dimensional robots. They have also paved the way for the development of planners for problems beyond basic path planning. Because of space limitations, this chapter concentrates on the above planners and some of their variants, and does not include a comprehensive description of all effective sampling-based planning methods.

Characteristics of Sampling-Based Planners

An important characteristic of the planners described in this chapter is that they do not attempt to explicitly construct the boundaries of the configuration space obstacles or represent cells of $\mathcal{Q}_{\text{free}}$. Instead, they rely on a procedure that can decide whether a given configuration of the robot is in collision with the obstacles or not. In some sense, sampling-based planners have very limited access to the configuration space. Efficient collision detection procedures ease the implementation of sampling-based planners and increase the range of their applicability. Furthermore, since collision detection is a separate module, it can be tailored to specific robots and applications. Recent advances in collision detection algorithms have contributed heavily to the success of sampling-based planners. Any future performance improvements in collision checking, which is an active area of research, will also benefit directly the performance of sampling-based planners. Examples of available collision detection packages include GJK [89, 163], SOLID [420, 421], V-Clip [316], I-Collide [115, 290], V-Collide [199], QuickCD [238], PQP [261], RAPID [168], SWIFT [140], SWIFT++ [141], and others [88, 296, 357, 376].


Another important characteristic of sampling-based planners is that they can achieve some form of completeness. Completeness requires that the planner always answers a path-planning query correctly, in asymptotically bounded time. Complete planners cannot be implemented in practice for robots with more than three degrees of freedom due to their high combinatorial complexity. A weaker, but still interesting, form of completeness is the following: if a solution path exists, the planner will eventually find it. If the sampling of the sampling-based planner is random, then this form of completeness is called probabilistic completeness. If the sampling is deterministic, including quasirandom or sampling on a grid, this form of completeness is called resolution completeness with respect to the sampling resolution. Probabilistic completeness was shown for one of the

earliest sampling-based planners, called the Randomized Path Planner (RPP) [39,257], setting a standard for sampling-based methods. PRM was also shown to be probabilistically complete [195, 221-223, 228, 252]. The analysis of the probabilistic completeness for the basic PRM planner [221,228] is presented in this chapter. The theoretical results relate the probability that PRM fails to find a path to the running time of the planner. Hence there is not only experimental evidence that PRM planners work well; there is also theoretical evidence of why this is the case. The analysis also sheds light on why the basic PRM planner works well on a large class of difficult problems.

Overview of This Chapter

[Section 7.1](#) introduces PRM . In its basic form, PRM constructs a roadmap that represents the connectivity of Q_{free} . This roadmap can be used for answering multiple queries. Guidelines for the efficient implementation of this planner for a general robot are also given. The guidelines are also relevant for the efficient implementation of the other sampling-based planners described in this chapter. A number of different sampling methods and connection strategies for PRM are then presented. Planners that are optimized for single-queries are described in [section 7.2](#). In general, these planners generate trees in Q_{free} . Some of the most efficient single-query planners, such as EST and RRT planners, perform a conditional sampling of Q_{free} the samples generated depend on the currently constructed tree and the goal configuration. In [section 7.2](#) the EST and RRT planners are described in detail. The combination of the different sampling and connection strategies of [sections 7.1](#) and [7.2](#) leads to an even more powerful planner, SRT , which is described in [section 7.3](#). An analysis of PRM is given in [section 7.4](#). Various extensions of the generalized mover's problem are then discussed in [section 7.5](#), including applications from computational structural biology.

 [Previous](#)

[Next](#) 




Chapter 7 - Sampling-Based Algorithms

Principles of Robot Motion: Theory, Algorithms, and Implementation

by Howie Choset et al.

The MIT Press © 2005

 Previous

Next 

7.1 Probabilistic Roadmaps

The PRM planner is described in [231]. The planner resulted from the work of independent groups [225,226,344,345,404] and was further developed in [221,223,227,228]. PRM divides planning into two phases: the learning phase, during which a roadmap in $\mathcal{Q}_{\text{free}}$ is built; and the query phase, during which user-defined query configurations are connected with the precomputed roadmap. The nodes of the roadmap are configurations in $\mathcal{Q}_{\text{free}}$ and the edges of the roadmap correspond to free paths computed by a local planner. The objective of the first phase is to capture the connectivity of $\mathcal{Q}_{\text{free}}$ so that path-planning queries can be answered efficiently.

The basic PRM algorithm presented below can be used to solve high-dimensional problems such as the one in figure 7.1. It has been shown to be probabilistically complete [221,229,252]. In this section, the choices for the sampling and connection strategies of PRM are reduced to a bare minimum to facilitate the presentation. The emphasis here is to describe a planner that is easy to implement and works well even with rather high-dimensional problems (5-12 degrees of freedom).

7.1.1 Basic PRM

The basic PRM algorithm first constructs a roadmap in a probabilistic way for a given workspace. The roadmap is represented by an undirected graph $G = (V, E)$. The nodes in V are a set of robot configurations chosen by some method over $\mathcal{Q}_{\text{free}}$. For the moment, assume that the generation of configurations is done randomly from a uniform distribution. The edges in E correspond to paths; an edge (q_1, q_2) corresponds to a collision-free path connecting configurations q_1 and q_2 . These paths, which are referred to as local paths, are computed by a local planner. In its simplest form, the local planner connects two configurations by the straight line in $\mathcal{Q}_{\text{free}}$, if such a line exists.

In the query phase, the roadmap is used to solve individual path-planning problems. Given an initial configuration q_{init} and a goal configuration q_{goal} , the method first tries to connect q_{init} and q_{goal} to two nodes q' and q'' , respectively, in V . If successful, the planner then searches the graph G for a sequence of edges in E connecting q' to q'' . Finally, it transforms this sequence into a feasible path for the robot by recomputing the corresponding local paths and concatenating them. Local paths can be stored in the roadmap but this would increase the storage requirements of the roadmap, a topic which is discussed later in this section.

The roadmap can be reused and further augmented to capture the connectivity of $\mathcal{Q}_{\text{free}}$. Although the learning phase is usually performed before any path-planning query, the two phases can also be interwoven. It is reasonable to spend a considerable amount of time in the learning phase if the roadmap will be used to solve many queries.

Roadmap Construction

To make the presentation more precise, let

- Δ be the local planner that on input $(q, q') \in \mathcal{Q}_{\text{free}} \times \mathcal{Q}_{\text{free}}$ returns either a collisionfree path from q to q' or NIL if it cannot find such a path. Assume for the moment that Δ is symmetric and deterministic.
- dist be a function $\mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{R}^+ \cup \{0\}$, called the *distance function*, usually a metric on \mathcal{Q} .

Algorithm 6 describes the steps of the roadmap construction. For all algorithms described in this chapter, it should be noted that only the main steps are given and that implementation details are missing.

Initially, the graph $G = (V, E)$ is empty. Then, repeatedly, a configuration is sampled from \mathcal{Q} . For the moment, assume that the sampling is done according to a uniform random distribution on \mathcal{Q} . If the configuration is collision-free, it is added to the roadmap. The process is repeated until n collision-free configurations have been sampled. For every node $q \in V$, a set N_q of k closest neighbors to the configuration q according to some metric dist is chosen from V . The local planner is called to connect q to each node $q' \in N_q$. Whenever Δ succeeds in computing a feasible path between q and q' , the edge (q, q') is added to the roadmap. **Figure 7.3** shows a roadmap constructed for a point robot in a two-dimensional Euclidean workspace, where Δ is a straight-line planner.

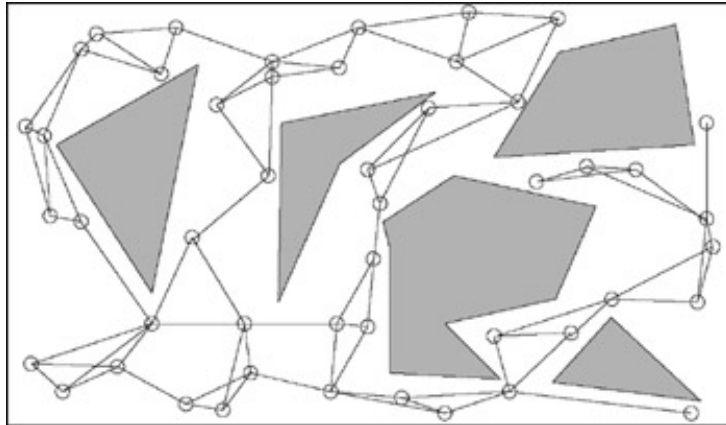


Figure 7.3: An example of a roadmap for a point robot in a two-dimensional Euclidean space. The gray areas are obstacles. The empty circles correspond to the nodes of the roadmap. The straight lines between circles correspond to edges. The number of k closest neighbors for the construction of the roadmap is three. The degree of a node can be greater than three since it may be included in the closest neighbor list of many nodes.

Algorithm 6: Roadmap Construction

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

1: $V \leftarrow \square$

2: $E \leftarrow \square$

3: **while** $|V| < n$ **do**

4: **repeat**

5: $q \leftarrow$ a random configuration in Q

6: **until** q is collision-free

7: $V \leftarrow V \cup \{q\}$

8: **end while**

9: **for all** $q \in V$ **do**

10: $N_q \leftarrow$ the k closest neighbors of q chosen from V according to $dist$

11: **for all** $q' \in N_q$ **do**

12: **if** $(q, q') \notin E$ **and** $\Delta(q, q') \neq \text{NIL}$ **then**

13: $E \leftarrow E \cup \{(q, q')\}$

14: **end if**

15: **end for**

16: **end for**

A number of components in [algorithm 6](#) are still unspecified. In particular, it needs to be defined how random configurations are created in line (5), how the closest neighbors are computed in line (10), how the distance function $dist$ used in line (10) is chosen, and how local paths are generated in line (12).

Query Phase

During the query phase, paths are found between arbitrary input configurations q_{init} and q_{goal} using the roadmap constructed in the learning phase. [Algorithm 7](#) illustrates this process.

Assume for the moment that Q_{free} is connected and that the roadmap consists of a single connected component. The main question is how to connect q_{init} and q_{goal} to the roadmap. Queries should terminate as quickly as possible, so an inexpensive algorithm is desired here. The strategy used in [algorithm 7](#) to connect q_{init} to the roadmap is to consider the k closest nodes in the roadmap in order of increasing distance from q_{init} , according to the metric $dist$, and try to connect q_{init} to each of them with the local planner until one connection succeeds. The number of closest neighbors considered in [algorithm 7](#) can be different from the one in [algorithm 6](#). The same procedure is used to connect q_{goal} to the roadmap.

If the connection of q_{init} and q_{goal} to the roadmap is successful, the shortest path is found on the roadmap between q_{init} and q_{goal} according to $dist$ (e.g., using Dijkstra's algorithm or the

A* algorithm). If one wishes, this path may be improved by running a smoothing postprocessing algorithm. Figure 7.4 shows the solution to a query solved with the roadmap from figure 7.3.

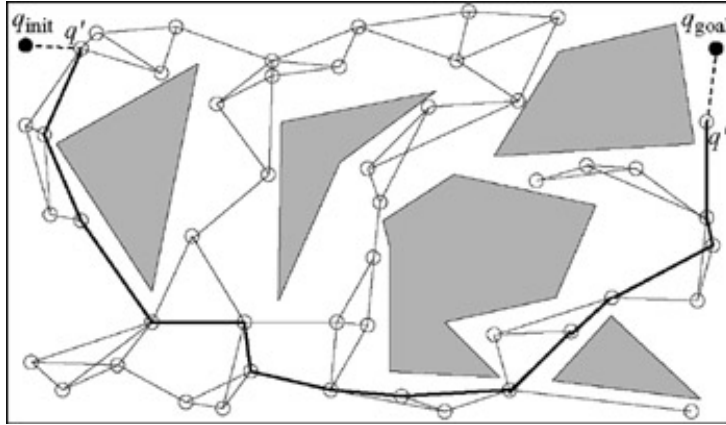


Figure 7.4: An example of how to solve a query with the roadmap from figure 7.3. The configurations q_{init} and q_{goal} are first connected to the roadmap through q' and q'' . Then a graph-search algorithm returns the shortest path denoted by the thick black lines.

In general, the roadmap may consist of several connected components. This is very likely when \mathcal{Q}_{free} is itself not connected, but it may also happen when \mathcal{Q}_{free} is connected, and the roadmap has not managed to capture the connectivity of \mathcal{Q}_{free} . If the roadmap contains several components, algorithm 7 can be used to connect both q_{init} and q_{goal} to two nodes in the same connected component of the roadmap, e.g., by giving it as input a single connected component of G . All components of G should be considered. If the connection of q_{init} and q_{goal} to the same connected component of the roadmap succeeds, a path is constructed as in the single-component case. The method returns failure if it cannot connect both q_{init} and q_{goal} to the same roadmap component.

Algorithm 7: Solve Query Algorithm

Input:

q_{init} : the initial configuration

q_{goal} : the goal configuration

k : the number of closest neighbors to examine for each configuration

$G = (V, E)$: the roadmap computed by algorithm 6

Output:

A path from q_{init} to q_{goal} or failure

1: $N_{q_{init}} \leftarrow$ the k closest neighbors of q_{init} from V according to $dist$

2: $N_{q_{goal}} \leftarrow$ the k closest neighbors of q_{goal} from V according to $dist$

3: $V \leftarrow \{q_{init}\} \cup \{q_{goal}\} \cup V$

4: set q' to be the closest neighbor of q_{init} in $N_{q_{init}}$

5: **repeat**

6: **if** $\Delta(q_{init}, q') \neq \text{NIL}$ **then**

7: $E \leftarrow (q_{init}, q') \cup E$

```

8:   else
9:     set  $q'$  to be the next closest neighbor of  $q_{\text{init}}$  in  $N_{q_{\text{init}}}$ 
10:  end if
11: until a connection was succesful or the set  $N_{q_{\text{init}}}$  is empty
12: set  $q'$  to be the closest neighbor of  $q_{\text{goal}}$  in  $N_{q_{\text{goal}}}$ 
13: repeat
14:   if  $\Delta(q_{\text{goal}}, q') \neq \text{NIL}$  then
15:      $E \leftarrow (q_{\text{goal}}, q') \cup E$ 
16:   else
17:     set  $q'$  to be the next closest neighbor of  $q_{\text{goal}}$  in  $N_{q_{\text{goal}}}$ 
18:   end if
19: until a connection was succesful or the set  $N_{q_{\text{goal}}}$  is empty
20:  $P \leftarrow \text{shortest path}(q_{\text{init}}, q_{\text{goal}}, G)$ 
21: if  $P$  is not empty then
22:   return  $P$ 
23: else
24:   return failure
25: end if

```

Adding to the Roadmap

If path-planning queries fail frequently, the roadmap may not adequately capture the connectivity of $\mathcal{Q}_{\text{free}}$. When this occurs, the current roadmap can be extended by resuming the construction step algorithm (exclude lines (1) and (2) from [algorithm 6](#) and pass as a parameter the current roadmap). It should be emphasized again that in this section we present a very basic PRM. It has been observed for example, that when trying to connect components biased sampling may be particularly effective [231]. Biased sampling (see Connection Sampling in [section 7.1.3](#)) increases the sampling density in areas of $\mathcal{Q}_{\text{free}}$ that have good chances to facilitate component connection.

Directed Roadmaps and Roadmaps That Store Local Paths

So far, it has been assumed that Δ is symmetric and deterministic. It is also possible to use a local planner Δ that is neither symmetric nor deterministic.

In many cases, connecting some configuration q to some configuration q' does not necessarily imply that the opposite can be done. If the local planner takes the robot from q to q' and the robot can also execute the path in reverse to go from q' to q , the roadmap is an undirected graph. Adding the edge (q, q') implies that the edge (q', q) can also be added. If local paths cannot be reversed, a directed roadmap must be constructed. A separate check must be performed to determine if the edge (q', q) can also be added to the roadmap.

A deterministic local planner will always return the same path between two configurations and the roadmap does not have to store the local path between the two configurations in the corresponding edge. The path can be recomputed if needed to answer a query. On the other hand, if a nondeterministic local planner is used, the roadmap will have to associate with

each edge the local path computed by Δ . In general, the use of nondeterministic local planners increases the storage requirements of the roadmap. It permits, however, the use of more powerful local planners, which can be an advantage in certain cases as discussed in [section 7.3](#).

7.1.2 A Practical Implementation of Basic PRM

One of the advantages of the basic PRM algorithm presented in the [previous section](#) is that it is easy to implement and performs well for a variety of problems. This section focuses on the details of a successful implementation of basic PRM that scales well for robots with many degrees of freedom. Issues that relate to a practical implementation of a planner, such as smoothing of the final path, are also discussed. These issues pertain to all planners in this chapter. The reader is also referred to [246] for details on implementation details and potential pitfalls.

Sampling Strategy: Uniform Distribution

In basic PRM [231] the nodes of the roadmap constitute a uniform random sampling of Q_{free} . To obtain a configuration, each translational degree of freedom can be drawn from the interval of allowed values of the corresponding degree of freedom using the uniform probability distribution over this interval. The same principle applies to rotational degrees of freedom but care should be taken not to favor specific orientations because of the representation used (see the example at the end of [section 7.1.2](#) and [246]). The main idea is that the sampling distribution should be symmetry invariant. The sampled configuration is checked for collision. If it is collision-free, the sample is added to the nodes of the roadmap; otherwise, it is discarded. Collision checking can be done using a variety of existing general techniques, as mentioned above.

Sampling from a uniform distribution is the simplest method for generating sample configurations, but other methods could be used, as we describe below. [Section 7.4](#) offers a theoretical explanation of why sampling from a uniform distribution works well for many problems.

Connection Strategy: Selecting Closest Neighbors

Another important choice to be made is that of selecting the set N_q of closest neighbors to a configuration q . Many data structures have been proposed in the field of computational geometry that deal with the problem of efficiently calculating the closest neighbors to a point in a d -dimensional space. A relatively efficient method both in terms of space and time is the kd-tree data structure [124].

A d -dimensional kd-tree uses as input a set S of n points in d dimensions and constructs a binary tree that decomposes space into cells such that no cell contains too many points. A kd-tree is built recursively by splitting S by a plane into two subsets of roughly equal size: S_l , which includes points of S that lie to the left of the plane; and S_r , which includes the remaining points of S . The plane is stored at the root, and the left and right child are recursively constructed with input S_l and S_r , respectively. [Figure 7.5](#) illustrates the construction of a 2-dimensional kd-tree for ten points on a plane.

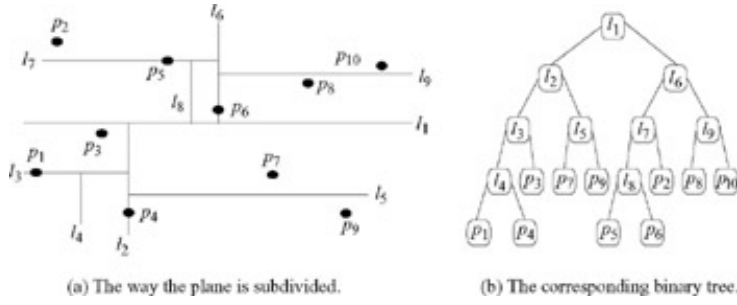


Figure 7.5: A kd-tree for ten points on a plane.

A kd-tree for a set of n points in d dimensions uses $O(dn)$ storage and can be built in $O(dn \log n)$ time. A rectangular range query takes $O(n^{1-\frac{1}{d}} + m)$ time, where m is the number of reported neighbors. As d grows large, the cost of using kd-trees becomes linear. The rectangular range query time can be reduced considerably by introducing a small approximation error. This modified approach is called Approximate Nearest Neighbor queries (ANN) and is becoming increasingly popular [30].

Distance Functions and Embeddings

Function $dist$ is used to resolve the k closest neighbors query. It should be defined so that, for any pair (q', q'') of configurations, $dist(q', q'')$ reflects the likelihood that the local planner will fail to compute a collision-free path between these configurations. One possibility is to define $dist(q', q'')$ as some measure of the workspace region swept by the robot, such as the area or the volume, when it moves in the absence of obstacles along the path $\Delta(q', q'')$. Intuitively, minimizing the swept volume, will minimize the chance of collision with the obstacles. An exact computation of swept areas or volumes is notoriously difficult, which is why heuristic metrics generally attempt to approximate the swept-volume metric (see [19,246]).

An approximate and inexpensive measure of the swept-region can be constructed as follows. The robot's configurations q' and q'' can be mapped to points in a Euclidean space, $emb(q')$ and $emb(q'')$, respectively, and the Euclidean distance between them can be used, i.e.,

$$dist(q', q'') = \| emb(q') - emb(q'') \| .$$

A practical choice for the embedding function is to select $p > 0$ points on the robot, concatenate them, and create a vector whose dimension is p multiplied by the dimension of the workspace of the robot. In order to represent a configuration q in the embedded space, the set of transformations corresponding to this configuration is applied to the p points, and $emb(q)$ is obtained. Distances can be easily defined using the equation above. An example is given at the very end of this section. Note, however, that this choice of embeddings has its shortcomings. In particular, it is not clear what the number p should be. It is also not clear how to choose p points so that the exact shape of the robot is taken into account. Furthermore, as is the case with the swept-volume metric, the embedding does not take into account obstacles. So even when two configurations are close to one another, connecting them may be impossible due to obstacles.

For the case of rigid body motion, an alternative solution is to split $dist$ into two components,

one that expresses the distance between two configurations due to translation and one due to orientation. For example, if X and R represent the translation and rotation components of the configuration $q = (X, R) \in SE(3)$ respectively, then

$$dist(q', q'') = w_t ||X' - X''|| + w_r f(R', R'')$$

is a weighted metric with the translation component $||X' - X''||$ using a standard Euclidean norm, and the positive scalar function $f(R', R'')$ returning typically an approximate measure of the distance between the rotations $R', R'' \in SO(3)$. The rotation distance is scaled relative to the translation distance via the weights w_t and w_r . A reasonable choice of $f(R', R'')$ is the length of the geodesic curve between R' and R'' . The selection of an appropriate rotation distance function $f(R', R'')$ depends on the representation for the orientation of the robot, such as Euler angles or quaternions.

One of the difficulties with this method is deciding proper weight values. Furthermore, the extension to articulated bodies is not straightforward. A thorough discussion of metrics for rigid body planning is given in [246].

The choices for the embedding, its dimensionality, and the *dist* can have a great effect on the efficiency of the PRM algorithm. Different problems may require different approaches and there is great interest in the motion-planning community in finding appropriate metrics [19,246] and embeddings for interesting instances of the generalized mover's problem.

Local Planner

In [section 7.1](#), it was assumed that is symmetric and deterministic. This is a design decision and it is possible to accommodate planners that are nondeterministic, and/or not symmetric.

Another important design decision is related to how fast the local planner should be. There is clearly a tradeoff between the time spent in each individual call of this planner and the number of calls. If a powerful local planner is used, it would often succeed in finding a path when one exists. Hence, relatively few nodes might be required to build a roadmap

capturing the connectivity of \mathcal{Q}_{free} sufficiently well to reliably answer path-planning queries. Such a local planner would probably be rather slow, but this could be somewhat compensated by the small number of calls needed. On the other hand, a very fast planner is likely to be less successful. It will require more configurations to be included in the roadmap and as a result, the local planner is called more times for the connections between nodes. Each call will be cheaper, however. In [section 7.3](#), a roadmap technique that uses a powerful local planner is discussed.

The choice of the local planner also affects the query phase. It is important to be able to connect any given q_{init} and q_{goal} configurations to the roadmap or to detect very quickly that no such connection is possible. This requires that the roadmap be dense enough that it always contains at least some nodes to which it is easy to connect q_{init} and q_{goal} . It thus seems preferable to use a very fast local planner, even if it is not too powerful, and build large roadmaps with configurations widely distributed over \mathcal{Q}_{free} . In addition, if the local planner is very fast, the same planner can be used to connect q_{init} and q_{goal} to the roadmap at query time. Discussions of the use of different local planners can be found in [14,162,203,221].

One popular planner, applicable to all holonomic robots, connects any two given configurations by a straight-line segment in \mathcal{Q} and checks this line segment for collision. Care should be taken to interpolate the translation and rotation components separately (see [246]). There are two commonly-used choices for collision checking, the incremental and the subdivision collision-checking algorithms. In both cases, the line segment, or more generally, any path generated by the local planner between configurations q' and q'' , is discretized into a number of configurations (q_1, \dots, q_ℓ) , where $q' = q_1$ and $q'' = q_\ell$. The distance between any two consecutive configurations q_i and q_{i+1} is less than some positive constant `step_size`. This value is problem specific and is defined by the user. It is important to note that again sampling is used to determine if a local path is collision-free. But in this case, sampling is done at a much finer level than was done for node generation and this is a very important feature of PRM. In general, the value of `step_size` needs to be very small to guarantee that all collisions are found.

In the case of incremental collision checking, the robot is positioned at q' and moved at each step by `step_size` along the straight line in \mathcal{Q} between q' and q'' . A collision check is performed at the end of each step. The algorithm terminates as soon as a collision is detected or when q'' is reached.

In the case of the subdivision collision checking, the middle point q_m of the straight line in \mathcal{Q} between q' and q'' is first checked for collision. Then the algorithm recurses on the straight lines between (q', q_m) and (q_m, q'') . The recursion halts when a collision is found or the length of the line segment is less than `step_size`.

In both algorithms, the path is considered collision-free if none of the intermediate configurations yields collision. Neither algorithm has a clear theoretical advantage over the other, but in practice the subdivision collision checking algorithm tends to perform better [162, 367]. The reason is that, in general, shorter paths tend to be collision-free. Subdivision collision checking cuts down the length of the local path as soon as possible. It is also possible to use an adaptive subdivision collision-checking algorithm that dynamically adjusts `step_size`. In [376], `step_size` is determined by relating the distance between the robot and the workspace obstacles to the maximum length of the path traced out by any point on the robot. Furthermore, the method in [376] is exact, i.e., it always finds a collision when a collision exists, whereas the above discretization techniques may miss a collision if `step_size` is too large.

Figure 7.6 illustrates how the incremental and subdivision collision-checking algorithms are sampling the straight line between two configurations q' and q'' . In this example, the subdivision algorithm performs a smaller number of collision checks. If the obstacle had been close to q' , then the incremental algorithm would have performed a smaller number of collision checks.

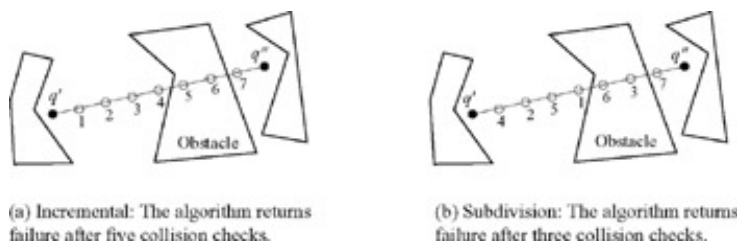


Figure 7.6: Sampling along the straight line path between two configurations q' and q'' .

The numbers correspond to the order in which each strategy checks the samples for collision.

Postprocessing Queries

A postprocessing step may be applied to the path connecting q_{init} to q_{goal} to improve its quality according to some criteria. For example, shortness and smoothness might be desirable. Postprocessing is applicable to any path-planning algorithm, but is presented here for completeness of the implementation guidelines of the basic PRM.

From a given path, a shorter path could be obtained by checking whether nonadjacent configurations q_1 and q_2 along the path can be connected with the local planner. This idea has been described often in the literature (e.g., [150, 383]). The points q_1 and q_2 could be chosen randomly. Another alternative would be a greedy approach. Start from q_{init} and try to connect directly to the target q_{goal} . If this step fails, start from the configuration after q_{init} and try again. Repeat until a connection can be made to q_{goal} , say from the point q_0 . Now set the target to q_0 and begin again, trying to connect from q_{init} to q_0 , and repeat the procedure. This procedure can also be applied toward the opposite direction. Figure 7.7 illustrates the application of the greedy approach in the forward direction to shorten a path in a two-dimensional Euclidean workspace.

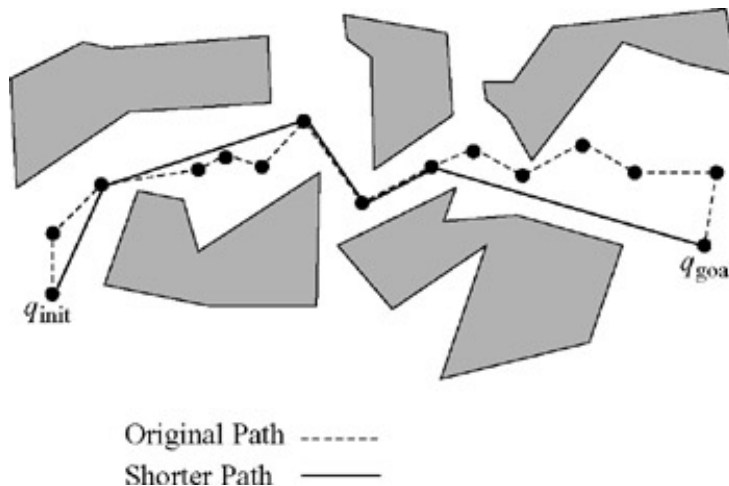


Figure 7.7: Processing the path returned from PRM to get a shorter path with the greedy approach.

There are various reasons why configurations q_1 and q_2 along a path may have not been connected with an edge from the roadmap construction step of PRM. They may not be close according to the distance function $dist$, and the k closest neighbor query may not return them as neighbors. They may, however, be in a relatively uncluttered part of Q_{free} and a long edge connecting them may still be possible. These cases will occur more frequently if the Creating Sparse Roadmaps connection strategy has been used (see section 7.1.4).

Instead of shortening the path, a different objective may be to get a path with smooth curvature. A possible approach to this is to use interpolating curves, such as splines, and use the configurations that have been computed by PRM as the interpolation points for the curves. In this case, collision checking is performed along the curves until curves that satisfy

both the smoothness properties and the collision avoidance criteria are found.

Postprocessing steps such as path shortening and path smoothing can improve the quality of the path, but can also impose a significant overhead on the time that it takes to report the results of a query. In general, if paths with certain optimality criteria are desired, it is worth trying to build these paths during the roadmap construction phase of PRM. For example, a large dense roadmap will probably yield shorter paths than a smaller and sparser roadmap.

An Example

Figure 7.8(a) shows a motion-planning problem for a robot in a three-dimensional workspace. The robot is a rigid nonconvex polyhedral object; it can freely translate and rotate in the workspace as long as it does not collide with the obstacles. The workspace is made up of a rigid thin wall that has a narrow passage. A bounding box is defined that contains the wall and is small enough so that it does not allow the robot to move from one side of the wall to the other without going through the narrow passage. The goal is to build a roadmap that a planner can use to successively solve motion-planning queries where q_{init} and q_{goal} appear on the two different sides of the wall.

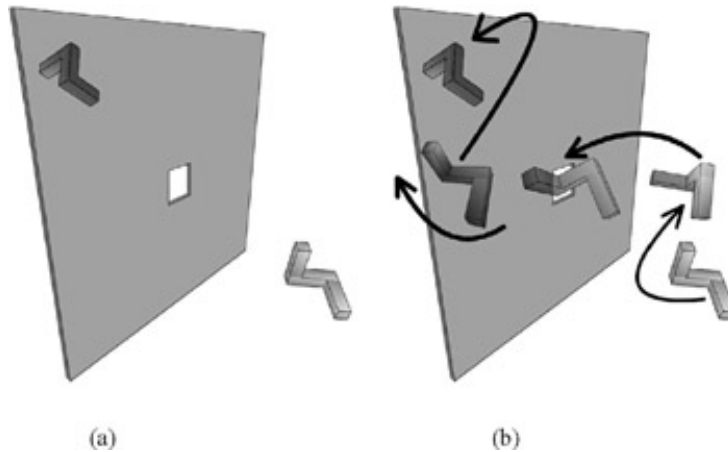


Figure 7.8: An example of a motion-planning problem where both the robot and the obstacles are a collection of polyhedral objects in three dimensions. Parts of the robot on the other side of the wall are indicated by the darker color. (a) The initial and goal configuration of the query. (b) A path produced from a PRM with $n = 1000$ and $k = 10$.

The problem has six degrees of freedom, three translational and three rotational. The configuration $q = (p, r)$ of the robot can be represented by a point p expressing the translational component and a quaternion r (see [appendix E](#)) expressing the rotational component. A configuration is generated by picking at random a sample from a uniform distribution from a subset of allowable positions in \mathbb{R}^3 and picking a random axis of rotation and a random angle for the quaternion (for details see [246]).

In order to find the k closest neighbors of a configuration, configurations are embedded in a space where Euclidean distance is defined. A method that works well in practice is to choose a pair of points on the surface of the robot that have maximum distance and construct a six-dimensional vector $\text{emb}(q)$ for the robot's initial configuration. If q' is obtained by applying a translation and rotation transformation to q , then $\text{emb}(q')$ is obtained by

applying the same transformations to the pair of points in $\text{emb}(q)$. The distance metric dist is then defined as the Euclidean distance of the two embeddings.

For every configuration and its k closest counterparts, the subdivision collision-checking algorithm is used to check if the straight line in \mathcal{Q} is collision-free. Intermediate configurations between $q' = (p', r')$ and $q'' = (p'', r'')$ are obtained by performing linear interpolations on p' and p'' and spherical interpolations on r' and r'' . The edge (q', q'') is added to the roadmap when all the intermediate configurations are collision-free.

When the roadmap has been completed, it can be used to solve user-specified queries. The k closest neighbors for the query points are calculated and the local planner attempts to connect q_{init} and q_{goal} to them. As soon as they are connected to the same component, an A^* algorithm is run on the graph to find the path. Figure 7.8(b) shows intermediate configurations of a path returned by the above procedure.

7.1.3 PRM Sampling Strategies

Several node-sampling strategies have been developed over the years for PRM. For many path-planning problems, a surprisingly large number of general sampling schemes will provide reasonable results (see e.g., the comparison of sampling schemes given in [162]). The analysis of section 7.4 provides some insight as to why this is the case. Intuitively, many planning problems in the physical world are difficult but not "pathological" (as in the kind of problem one encounters in NP-hardness proofs). Without doubt, however, the choice of the node-sampling strategy can play a significant role in the performance of PRM. This was observed in the original PRM publications which suggested mechanisms to generate samples in a non-uniform way [231]. Increasing the density of sampling in some areas of the free space is referred to as importance sampling and has been repeatedly demonstrated to increase the observed performance of PRM. In this section we describe several node-sampling schemes.

The uniform random sampling used in early work in PRM is the easiest sampling scheme to implement. As a random sampling method, it has the advantage that, in theory, a malicious opponent cannot defeat the planner by constructing carefully crafted inputs. It has the disadvantage, however, that, in difficult planning examples, the running time of PRM might vary across different runs. Nevertheless, random sampling works well in many practical cases involving robots with a large number of degrees of freedom.

There exist cases where uniform random sampling has poor performance. Often, this is the result of the so-called narrow passage problem. If a narrow passage exists in $\mathcal{Q}_{\text{free}}$ and it is absolutely necessary to go through that passage to solve a query, a sampling-based planner must select a sample from a potentially very small set in order to answer the planning query. A number of different sampling methods have been designed with the narrow passage problem in mind and are described below. The narrow passage problem still remains a challenge for PRM planners and is an active area of research.

The remainder of this section describes sampling strategies that have been developed with the narrow passage problem in mind and then other general sampling strategies. We conclude the section with a brief discussion of how one might select an appropriate sampling scheme for a particular problem.

Sampling Near the Obstacles

Obstacle-based sampling methods sample near the boundary of configuration-space obstacles. The motivation behind this kind of sampling is that narrow passages can be considered as thin corridors in $\mathcal{Q}_{\text{free}}$ surrounded by obstacles.

OBPRM [18] is one of the first and very successful representatives of obstacle-based sampling methods. Initially, OBPRM generates many configurations at random from a uniform distribution. For each configuration q_{in} found in collision, it generates a random direction v , and the planner finds a free configuration q_{out} in the direction v . Finally, it performs a simple binary search to find the closest free configuration q to the surface of the obstacle. Configuration q is added to the roadmap, while q_{in} and q_{out} are discarded.

The Gaussian sampler [59] addresses the narrow passage problem by sampling from a Gaussian distribution that is biased near the obstacles. The Gaussian distribution is obtained by first generating a configuration q_1 randomly from a uniform distribution. Then a distance step is chosen according to a normal distribution to generate a configuration q_2 at random at distance step from q_1 . Both configurations are discarded if both are in collision or if both are collision-free. A sample is added to the roadmap if it is collision-free and the other sample is in collision.

In [194], samples are generated in a dilated $\mathcal{Q}_{\text{free}}$ by allowing the robot to penetrate by some small constant distance into the obstacles. The dilation of $\mathcal{Q}_{\text{free}}$ widens narrow passages, making it easier for the planner to capture the connectivity of the space. During a second stage, all samples that do not lie in $\mathcal{Q}_{\text{free}}$ are pushed into $\mathcal{Q}_{\text{free}}$ by performing local resampling operations.

Sampling Inside Narrow Passages

The bridge planner [193] uses a bridge test to sample configurations inside narrow passages. In a bridge test, two configurations q' and q'' are sampled randomly from a uniform distribution in \mathcal{Q} . These configurations are considered for addition to the roadmap, but if they are both in collision, then the point q_m halfway between them is added to the roadmap if it is collision free. This is called a bridge test because the line segment between q' and q'' resembles a bridge with q' and q'' inside obstacles acting as piers and the midpoint q_m hovering over $\mathcal{Q}_{\text{free}}$. Observe that the geometry of narrow passages makes the construction of short bridges easy, while in open space the construction of short bridges is difficult. This allows the bridge planner to sample points inside narrow passages by favoring the construction of short bridges.

An efficient solution to the narrow passage problem would generate samples that are inside narrow passages but as far away as possible from the obstacles. The Generalized Voronoi Diagrams (GVDs) described in [chapter 5](#) have exactly this property. Although exact computation of the GVD is impractical for high-dimensional configuration spaces, it is possible to find samples on the GVD without computing it explicitly. This can be done by a retraction scheme [427]. The retraction is achieved by a bisection method that moves each sample configuration until it is equidistant from two points on the boundary of $\mathcal{Q}_{\text{free}}$.

A simpler approach is to compute the GVD of the workspace and generate samples that somehow conform to this GVD [155,171,191]. For example, the robot can have some

predefined handle points (e.g., end-points of the longest diameter of the robot) and sampling can place those handle points as close to the GVD as possible with the hope of aligning the whole robot with narrow passages. The disadvantage of workspace-GVD sampling is that it is in general difficult to generate configurations of the robot close to the GVD (details are given in [155, 171, 191]). The advantage of workspace-GVD sampling is that the GVD captures well narrow passages in the workspace that typically lead to narrow passages in Q_{free} . Additionally, an approximation of the GVD of the workspace can be computed efficiently using graphics hardware [352] which is one of the reasons why this sampling method is popular for virtual walkthroughs and related simulations.

Visibility-Based Sampling

The goal of the visibility-based PRM [337] is to produce visibility roadmaps with a small number of nodes by structuring the configuration space into visibility domains. The visibility domain of a configuration q includes all configurations that can be connected to q by the local planner. This planner, unlike PRM which accepts all the free configurations generated in the construction stage, adds to the roadmap only those configurations q that satisfy one of two criteria: (1) q cannot be connected to any existing node, i.e., q is a new component, or (2) q connects at least two existing components. In this way, the number of configurations in the roadmap is kept small.

Manipulability-Based Sampling

Manipulability-based sampling [281, 282] is an importance-sampling approach that exploits the manipulability measure associated with the manipulator Jacobian [432]. Intuitively, manipulability characterizes the arm's freedom of motion for a given configuration. The motivation for using manipulability as a bias for sampling is as follows. In regions of the configuration space where manipulability is high, the robot has great dexterity, and therefore relatively fewer samples should be required in these areas. Regions of the configuration space where manipulability is low tend to be near (or to include) singular configurations of the arm. Near singularities, the range of possible motions is reduced, and therefore such regions should be sampled more densely.

Let $J(q)$ denote the manipulator Jacobian matrix (i.e., the matrix that relates velocities of the end effector to joint velocities). For a redundant arm (e.g., an arm with more than six joints for a 3D workspace) the manipulability in configuration q is given by

$$(7.1) \quad \omega(q) = \sqrt{\det J(q)J^T(q)}.$$

To bias sampling, an approximation to the cumulative density function (CDF) for ω is created. Samples are then drawn from a uniform density on the configuration space, and rejected with probability proportional to the associated CDF value of their manipulability value.

Quasirandom Sampling

A number of deterministic (sometimes called quasirandom) alternatives to random sampling have been used [62, 269, 291, 292]. These alternatives were first introduced in the context of Monte Carlo integration and aim to optimize various properties of the distribution of the samples. Before discussing some of these alternatives, we briefly describe two ways to

evaluate a set of samples.

Let P be a set of point samples on some space X , and N be the number of points in P . One way to evaluate the quality of the samples in P is to assess how "uniformly" the points in P cover X . This is done with respect to a specific collection of subsets of X , called a *range space*, denoted by \mathcal{R} . Let \mathcal{R} be the set of all axis-aligned rectangular subsets of X , and define μ to be the measure (or volume) of a set. Since P contains N points, the difference between the relative volumes of R to X and the fraction of samples contained in $R \in \mathcal{R}$ is given by

$$\left| \frac{\mu(R)}{\mu(X)} - \frac{|P \cap R|}{N} \right|.$$

If we take the supremum of this difference over all $R \in \mathcal{R}$ we obtain the concept of *discrepancy*.

DEFINITION 7.1.1

The *discrepancy* of point set P with respect to range space \mathcal{R} over some space X is defined as

$$D(P, \mathcal{R}) = \sup_{R \in \mathcal{R}} \left| \frac{\mu(R)}{\mu(X)} - \frac{|P \cap R|}{N} \right|.$$

It is not necessary to take \mathcal{R} as the subset of axis-aligned rectangles, but this choice gives an intuitive understanding of discrepancy. Another common choice is to take \mathcal{R} as the set of d -balls, i.e., for each $R \in \mathcal{R}$ we have $R = \{x' \mid \|x - x'\| < \epsilon\}$, for some point x and radius $\epsilon > 0$.

While discrepancy provides a measure of how uniformly points are distributed over the space X , *dispersion* provides a measure of the largest portion of X that contains no points in P . For a given metric ρ , the distance between a point $x \in X$ and a point $p \in P$ is given by $\rho(x, p)$. Thus, $\min_{p \in P} \rho(x, p)$ gives the distance from x to the nearest point in P . If we take ρ to be the Euclidean metric, this gives the largest empty ball centered on x . If we then take the minimization over all points in X , we obtain the size of the largest empty ball in X . This is exactly the concept of dispersion.

DEFINITION 7.1.2

The *dispersion* δ of point set P with respect to the metric ρ is given by

$$\delta(P, \rho) = \sup_{x \in X} \min_{p \in P} \rho(x, p).$$

An important result due to Sukharev gives a bound on the number of samples required to achieve a given dispersion. In particular, the *Sukharev sampling criterion* states that when p

is taken as the L_∞ norm, a set P of N samples on the d -dimensional unit cube will have

$$\delta(P, \rho) \geq \frac{1}{2 \lfloor N^{\frac{1}{d}} \rfloor}.$$

So, to achieve a given dispersion value, say δ^* , since N must be an integer, we have

$$\delta^* \geq \frac{1}{2 \lfloor N^{\frac{1}{d}} \rfloor} \rightarrow N \geq \left(\frac{1}{2\delta^*} \right)^d,$$

i.e., the number of samples required to achieve a desired dispersion grows exponentially with the dimension of the space. In some sense, this result implies that to minimize dispersion, sampling on a regular grid will yield results that are as good as possible.

Now that we have quantitative measures for the quality of a set of samples, we describe some common ways to generate samples. For the case of $X = [0, 1]$ the *Van der Corput sequence* gives a set of samples that minimizes both dispersion and discrepancy. The n^{th} sample in the sequence is generated as follows. Let $a_i \in \{0, 1\}$ be the coefficients that define the binary representation of n ,

$$n = \sum_i a_i 2^i = a_0 + a_1 2 + a_2 2^2 + \dots.$$

The n^{th} element of the Van der Corput sequence, $\Phi(n)$, is defined as

$$\Phi(n) = \sum_i a_i 2^{-(i+1)} = a_0 2^{-1} + a_1 2^{-2} + \dots.$$

Figure 7.9(a) shows the first sixteen elements of a Van der Corput sequence.

n	n (binary)	$\Phi(n)$ (binary)	$\Phi(n)$
0	0	0.0	0
1	1	0.1	1/2
2	10	0.01	1/4
3	11	0.11	3/4
4	100	0.001	1/8
5	101	0.101	5/8
6	110	0.011	3/8
7	111	0.111	7/8
8	1000	0.0001	1/16
9	1001	0.1001	9/16
10	1010	0.0101	5/16

11	1011	0.1101	13/16
12	1100	0.0011	3/16
13	1101	0.1011	11/16
14	1110	0.0111	7/16
15	1111	0.1111	15/16

n	$\Phi_2(n)$	$\Phi_1(n)$
0	0	0
1	1/3	1/2
2	2/3	1/4
3	1/9	3/4
4	4/9	1/8
5	7/9	5/8
6	2/9	3/8
7	5/9	7/8
8	8/9	1/16
9	1/27	9/16
10	10/27	5/16
11	19/27	13/16
12	4/27	3/16
13	13/27	11/16
14	22/27	7/16
15	7/27	15/16

Figure 7.9: (a) Van der Corput sequence, (b) Halton sequence for $d = 2$.

The Van der Corput sequence can only be used to sample the real line. The *Halton sequence* generalizes the Van der Corput sequence to d dimensions. Let $\{b_j\}$ define a set of d relatively prime integers, e.g., $b_1 = 2$, $b_2 = 3$, $b_3 = 5$, $b_4 = 7$, The integer n has a representation in base b_j given by

$$n = \sum_i a_{ij} b_j^i, \quad a_{ij} \in \{0, 1, \dots, b_j - 1\}$$

and $\Phi_{b_j}(n)$ is defined as

$$\Phi_{b_j}(n) = \sum a_{ij} b_j^{-(i+1)}.$$

The n^{th} sample is then defined by the coordinates $p_n = (\Phi_{b_1}(n), \Phi_{b_2}(n), \dots, \Phi_{b_d}(n))$. Figure 7.9(b) shows the first sixteen elements of a Halton sequence for $b_1 = 2, b_2 = 3$.

When the range space \mathcal{R} is a set of axis-aligned rectangular subsets of X , the discrepancy for the Halton sequence is bounded by

$$D(P, \mathcal{R}) \leq O\left(\frac{\log^d N}{N}\right).$$

When the range space \mathcal{R} is the set of d -balls, the discrepancy is bounded by

$$D(P, \mathcal{R}) \leq O\left(N^{-\frac{(d+1)}{2}}\right).$$

When N is specified, a *Hammersley sequence* (sometimes called a Hammersley point set, since the number of points is known and finite) achieves the best possible asymptotic discrepancy. The n^{th} point in a Hammersley sequence is obtained by using the first $d - 1$ coordinates of a point in the Halton sequence, with the ratio n/N as the first coordinate,

$$p_n = (n/N, \Phi_{b_1}(n), \Phi_{b_2}(n), \dots, \Phi_{b_{d-1}}(n)), \quad n = 0 \dots N - 1.$$

Figure 7.10 shows point sets generated using a random number generator (figure 7.10a), a Halton sequence (figure 7.10b), and a Hammersley sequence (figure 7.10c). Each point set contains 1024 points.

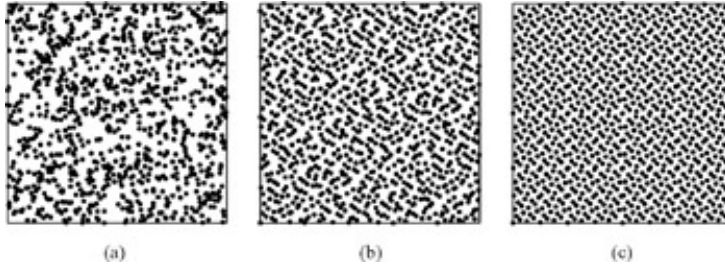


Figure 7.10: These figures show 1024 samples generated in the plane using (a) a random number generator, (b) a Halton sequence, (c) a Hammersley sequence.

The use of quasirandom sequences has the advantage that the running time is guaranteed to be the same for all the runs due to the deterministic nature of the point generation process. The resulting planner is resolution complete. The analysis of section 7.4 also sheds light as to why quasirandom sequences work well. As with any deterministic sampling method however, it is possible to construct examples where the performance of the planner deteriorates. As a remedy, it has been suggested to perturb the sequence [162]. The perturbation is achieved by choosing a random configuration from a uniform distribution in a small area around the sample point being added to the sequence. The area is gradually reduced as more points are added to the sequence. Certain quasirandom sequences can also be seen as generating points in a multiresolution grid in \mathcal{Q} [269].

Grid-Based Sampling

Grid-based planners have appeared in the early planning literature [244,274] but did not use some key abstractions of PRM such as the collision checking primitives. The nodes of a grid can be an effective sampling strategy in the PRM setting. Especially when combined with efficient node connection schemes (see section 7.1.4), they can result in powerful planners for problems arising in industrial settings [52]. A natural way of using grid-based search in a PRM is to use a rather coarse resolution for the grid and take advantage of the collision-checking abstraction; moving from one grid node q to a neighboring grid node q' would require collision checking, and hence sampling, at a finer resolution between the nodes. During the query phase, attempts are made to connect q_{init} and q_{goal} to nearby grid points. The resolution of the grid that is used to build the roadmap can be progressively increased either by adding points one at a time or by adding an entire hyperplane of samples chosen to fill the largest gap in the existing grid [52]. Of particular interest for path planning is the use of infinite sequences based on regular structures, which incrementally enhance their resolution. Recent work has demonstrated the use of such sequences for building lattices and other regular structures that have an implicit neighborhood structure, which is very useful for PRMs [269,291]. A grid-based path-planning algorithm is resolution complete.

Connection Sampling

Connection sampling [221, 231] generates samples that facilitate the connection of the roadmap and can be combined with all previously described sampling methods. Typically, if a small number of configurations is initially generated, there may exist a few disconnected components at the end of the construction step. If the roadmap under construction is disconnected in a place where $\mathcal{Q}_{\text{free}}$ is not, this place may correspond to some difficult area of $\mathcal{Q}_{\text{free}}$, possibly to a narrow passage of $\mathcal{Q}_{\text{free}}$. The idea underlying connection sampling is to select a number of configurations from the roadmap that are likely to lie in such regions and expand them. The expansion of a configuration q involves selecting a new free configuration in the neighborhood of q as described below, adding this configuration to the roadmap, and trying to connect it to other configurations of the roadmap in the same manner as in the construction step. The connection sampling step increases the density of the roadmap in regions of $\mathcal{Q}_{\text{free}}$ that are believed to be difficult. Since the gaps between components of the roadmap are typically located in these regions, the connectivity of the roadmap is likely to increase. Connection sampling thus never creates new components in the roadmap. At worst, it fails to reduce the number of components.

A simple probabilistic scheme can be used for connection sampling. Each configuration q is associated with a heuristic measure of the difficulty of the region around q expressed by a positive weight $w(q)$. Thus, $w(q)$ is large whenever q is considered to be in a difficult region. Weights are normalized so that their sum for all configurations in the roadmap is one. Then, repeatedly, a configuration q is selected from the roadmap with probability

$$Pr(q \text{ is selected}) = w(q),$$

and then q is expanded. The weights can be computed only once at the beginning of the process and not modified when new configurations are added to the roadmap, or can be modified periodically.

There are several ways to define the heuristic weight $w(q)$ [221,231]. A function that has

been found to work well in practice is the following. Let $\deg(q)$ be the number of configurations to which q is connected. Then,

$$w(q) = \frac{\frac{1}{\deg(q)+1}}{\sum_{q' \in V} \frac{1}{\deg(q')+1}}.$$

The expansion of a configuration q requires the generation of a configuration in the neighborhood of q . Typically, such a configuration can be found easily by selecting values for the degrees of freedom of the robot within a small interval centered at the values of the corresponding degrees of freedom of q . If this fails, a small random-bounce walk may be used to arrive at a new collision-free configuration. For holonomic robots, a random-bounce walk [231] from q consists of repeatedly picking at random a direction of motion and moving in this direction until an obstacle is hit. When a collision occurs, a new random direction is chosen. The above steps are repeated for a number of times. The configuration q' reached by the random-bounce walk and the edge (q, q') are inserted into the roadmap. Moreover, the path computed between q and q' is explicitly stored, since it was generated by a nondeterministic technique. The fact that q' belongs to the same connected component as q is also recorded. Then attempts are made to connect q' to the other connected components of the roadmap in the same way as in the construction step of PRM.

Choosing Among Different Sampling Strategies

Choosing among different sampling strategies is an open issue. Here, we give some very rough guidelines on how to choose a sampling strategy.

The success of PRM should be partly attributed to the fact that for a large range of problems (difficult but not "pathological" problems—see [section 7.4](#)) several simple sampling strategies work well. For example, uniform random sampling works well for many problems found in practice involving 3–7 degrees of freedom. If consistency in the running time is an issue, quasirandom sampling and lattice-based sampling provide some advantages. When the dimension grows, and again for problems that do not exhibit pathological behavior, random sampling is the simplest way to go. When problems that have narrow passages are considered, sampling-based strategies that were designed with narrow passages in mind should be used.

Combinations of different sampling methods are possible and in many cases critical for success. If π_A and π_B are two different sampling methods, a weighted hybrid sampling method π can be produced by setting $\pi = (1 - w)\pi_A + w\pi_B$. For example, connection sampling could be used in combination with random sampling [231] or OBPRM sampling. One sampling strategy can also be considered a filter for another. For example, the Gaussian sampler can be used to filter nodes created according to the bridge test [263].

None of the sampling methods described in this chapter provides clearly the best strategy across all planning problems. Sampling should also be considered in relation with the connection strategy used (see [section 7.1.4](#)) and the local planner used (see [14, 162, 203, 221] and [section 7.3](#)). Finally, it must be emphasized that it is possible to create "pathological" path-planning instances that will be arbitrarily hard for any sampling-based planner.

7.1.4 PRM Connection Strategies

An important aspect of PRM is the selection of pairs of configurations that will be tried for connections by a local planner. The objective is to select those configurations for which the local planner is likely to succeed. As has been discussed, one possible choice is to use the local planner to connect every configuration to all of its k closest neighbors. The rationale is that nearby samples lead to short connections that have good chances of being collision free. This section discusses some other approaches, their advantages and disadvantages. Clearly, the function used to select the neighbors and the implemented local planner can drastically affect the performance [19,246] of any connection strategy described in this section.

Creating Sparse Roadmaps

A method that can speed up the roadmap construction step is to avoid the computation of edges that are part of the same connected component [231,404]. Since there exists a path between any two configurations in a connected component, the addition of the new edge will not improve the connectivity of the roadmap. Several implementations of this idea have been proposed. The simplest is to connect a configuration with the nearest node in each component that lies close enough. This method avoids many calls to the local planner and consequently speeds up the roadmap construction step. As the graph is being built, the connected components can be maintained by using a fast disjoint-set data structure [119].

With the above method, no cycles can be created and the resulting graph is a forest, i.e., a collection of trees. Since a query would never succeed due to an edge that is part of a cycle, it is indeed sensible not to consume time and space computing and storing such an edge. In some cases, however, the absence of cycles may lead the query phase to construct unnecessarily long paths. This drawback can be mitigated by applying postprocessing techniques, such as smoothing, on the resulting path. It has been observed however that allowing some redundant edges to be computed during the roadmap construction phase (e.g., two or three per node) can significantly improve the quality of the original path without significant overhead [162]. Recent work shows how to add useful cycles in PRM roadmaps that result in higher quality (shorter) paths [336].

Connecting Connected Components

The roadmap constructed by PRM is aimed at capturing the connectivity of \mathcal{Q}_{free} . In some cases, due to the difficulty of the problem or the inadequate number of samples being generated, the roadmap may consist of several connected components. The quality of the roadmap can be improved by employing strategies aimed at connecting different components of the roadmap. Connection sampling, introduced in [section 7.1.3](#), attempts to connect different components of the roadmap by placing more nodes in difficult regions of \mathcal{Q}_{free} . [Section 7.2](#) describes sampling-based tree planners that can be very effective in connecting different components of the roadmap. This is exploited in the planner described in [section 7.3](#). Random walks and powerful planners such as RPP [40] can also be used to connect components [221]. Other strategies are described in [323].

Lazy Evaluation

The idea behind lazy evaluation is to speed up performance by doing collision checks only

when it is absolutely necessary. Lazy evaluation can be applied to almost all the sampling-based planners presented in this chapter [52-54]. In this section, lazy evaluation is described as a node connection scheme. It has also given rise to very effective planners that will be described in the [next section](#).

When lazy evaluation is employed, PRM operates on a roadmap G , whose nodes and paths have not been fully evaluated. It is assumed that all nodes and all edges of a node to its k neighbors are free of collisions. Once PRM is presented with a query, it connects q_{init} and q_{goal} to two close nodes of G . The planner then performs a graph search to find the shortest path between q_{init} and q_{goal} , according to the distance function used. Then the path is checked as follows. First, the nodes of G on the path are checked for collision. If a node is found in collision, it is removed from G together with all the edges originating from it. This procedure is repeated until a path with free nodes is discovered. The edges of that path are then checked. In order to avoid unnecessary collision checks, however, all edges along the path are first checked at a coarse resolution, and then at each iteration the resolution becomes finer and finer until it reaches the desired discretization. If an edge is found in collision, it is removed from G . The process of finding paths, checking their nodes and then checking their edges is repeated until a free path is found or all nodes of G have been visited. Once it is decided that a node of G is in \mathcal{Q}_{free} , this information is recorded to avoid future checks. For the edges, the resolution at which they have been checked for collision is also recorded so that if an edge is part of a future path, collision checks are not replicated. If no path is found and the nodes of G have been exhausted, new nodes and edges can be added to G . The new nodes can be sampled not only randomly but also from the difficult regions of \mathcal{Q}_{free} [54]. This kind of sampling is similar to the connection sampling strategy of PRM described in [section 7.1.3](#).

A related lazy scheme [335] assigns a probability to each edge of being collision free. This probability is computed by taking into account the resolution at which the edge has been checked. The edge probabilities can be used to search for a path in G that has good chances of being in \mathcal{Q}_{free} .

 [Previous](#)

[Next](#) 



Chapter 7 - Sampling-Based Algorithms

Principles of Robot Motion: Theory, Algorithms, and Implementation

by Howie Choset et al.

The MIT Press © 2005

 Previous

Next 

7.2 Single-Query Sampling-Based Planners

PRM was originally presented as a multiple-query planner: the goal was to create a roadmap that captures the connectivity of $\mathcal{Q}_{\text{free}}$ and then answer multiple user-defined queries very fast. In many planning instances, the answer to a single query is of interest and these instances are best served by single-query planners. Single-query planners attempt to solve a query as fast as possible and do not focus on the exploration of the entire $\mathcal{Q}_{\text{free}}$.

Many efficient single-query sampling-based planners exist. Some of them preceded PRM. One of the first widely used sampling-based planners was RPP [40]. RPP works by constructing potential fields over the workspace that attract control points of the robot to their corresponding positions in the goal configuration while pushing these robot points away from the obstacles (see also chapter 4). The workspace potentials are combined using an arbitration function to generate a configuration space potential. Starting from the initial configuration RPP performs a gradient motion until it reaches a local minimum. If the goal configuration has not been reached, RPP executes a series of random motions to escape the local minimum. In this way, RPP incrementally builds a graph of local minima, where the path joining two local minima is obtained by concatenating a random motion and a gradient descent motion. "Ariadne's clew" is another algorithm that uses samples in the configuration space [47,48]. The algorithm works by interleaving the exploration of \mathcal{Q} with searches for paths to the goal configuration. "Ariadne's clew" builds a tree from the initial configuration. During exploration, new configurations are placed in $\mathcal{Q}_{\text{free}}$ as far as possible from one another. The selection of configurations can be difficult and is done through genetic optimization. For each new configuration, a local search is performed to determine if the goal configuration is reachable from it. Many other algorithms (e.g., [33, 102, 165, 204]) explored the idea of planning by generating sample points in $\mathcal{Q}_{\text{free}}$, but will not be presented in this chapter due to space limitations. The planner in [204] called the ZZ-method bears some similarities with PRM.

PRM itself can also be used as single-query planner. In that case, q_{init} and q_{goal} should be inserted to the roadmap at the beginning. The planner should check periodically if the given query can be solved, that is if q_{init} and q_{goal} belong to the same component of the roadmap. At that point, the construction of the roadmap should be aborted. The sampling and connections strategies described in section 7.1 are all applicable here. In particular, the careful application of lazy evaluation has yielded an effective single-query PRM planner, which is called LazyPRM [52-54]. LazyPRM "creates" a roadmap whose nodes and edges have not been checked for collision. The planner performs a standard search to find a path from the initial to the goal configuration and starts checking the path for collisions as described in section 7.1.4. The planner stops when a collision-free path has been found and it was shown experimentally that this was achieved well before the roadmap was fully checked [53].

This section describes two planners that were designed primarily for single-query planning. The planners are Expansive-Spaces Trees (ESTs) [192, 195, 196, 235] and Rapidly-exploring Random Trees (RRTs) [249,270-272]. These planners also have the advantage that they are very efficient for kinodynamic planning (see section 7.5 and chapters 10, 11, and 12). For the moment, we concentrate on geometric path-planning.

ESTs and RRTs bias the sampling of configurations by maintaining two trees, T_{init} and T_{goal} , rooted at q_{init} and q_{goal} configurations, respectively, and then growing the trees toward each other until they are merged into one. It is possible to construct only a single tree rooted at q_{init} that grows toward q_{goal} , but, for geometric path-planning, this is usually less efficient than maintaining two trees. In the construction step, new configurations are sampled from $\mathcal{Q}_{\text{free}}$ near the boundaries of the two trees. A configuration is added to a tree only if it can be connected by the local planner to some existing configuration in the tree. In the merging step, a local planner attempts to connect pairs of configurations selected from both trees. If successful, the two trees become one connected component and a path from q_{init} to q_{goal} is returned.

For answering a single query, it is necessary to cover only the parts of Q_{free} relevant to the query. ESTs and RRTs developed sampling strategies that bias the sampling of the configurations toward the unexplored components of Q_{free} relevant to the query. The introduced sampling methods are fundamentally conditional: the generation of a new configuration depends on the initial and goal configuration and any previously generated configurations. The planners, however, are faced with the following dilemma: although it is important to search the part of Q_{free} that is relevant to the given query, the planners need to demonstrate that their sampling can potentially cover the whole Q_{free} . This is necessary for ensuring probabilistic completeness. ESTs are a purely forward projection/propagation method. An EST pushes the constructed tree to unexplored parts of Q_{free} by sampling points away from densely sampled areas. A rigorous analysis shows that Q_{free} will be covered under certain assumptions [192]. RRTs employ a steering strategy that pulls the tree to unexplored parts of Q_{free} . An RRT attempts to expand toward points in the free configuration space away from the tree. The algorithm has been shown to be probabilistically complete under certain assumptions [271]. Figure 7.11 shows a single tree expanded from q_{init} using a variant of EST [350].

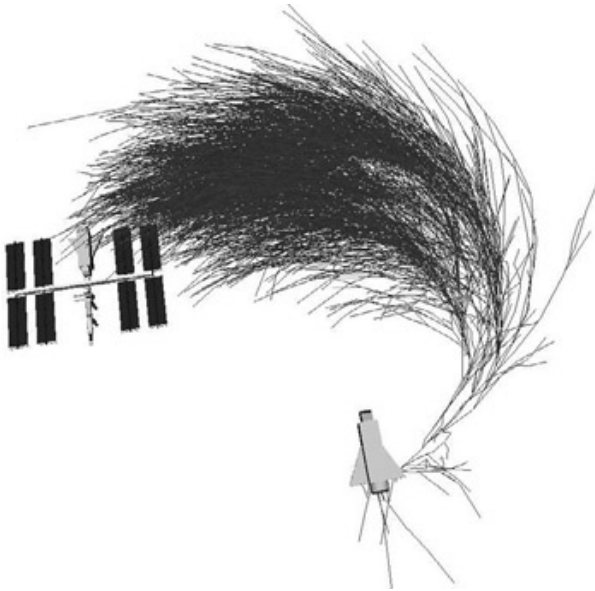


Figure 7.11: Tree generated by a tree-based motion planner for docking a space shuttle at the space station. (From Phillips and Kavraki [350].)

At the end of this section, the SBL [367] planner is described. SBL is a bi-directional EST that uses lazy evaluation for its node connection strategy. This allows the planner to explore the free space very efficiently and at the same time reduce the number of collision checks with further performance improvements over traditional ESTs.

7.2.1 Expansive-Spaces Trees

ESTs were initially developed as an efficient single-query planner that covers the space between q_{init} and q_{goal} rapidly [192, 195, 196, 235]. The developers of the algorithm did not use the acronym EST in their original publications. The acronym was later adopted and was inspired by the notion of "expansive" space used in the theoretical analysis of the algorithm. EST was initially geared toward kinodynamic problems, and for these problems a single tree is typically built (see section 7.5.1). A number of recent planners are based on or use ESTs [14, 350, 367]. The EST algorithm has been shown to be probabilistically complete [192].

Construction of Trees

Let T be one of the trees T_{init} or T_{goal} rooted at q_{init} and q_{goal} , respectively. The planner first selects a configuration q in T from which to grow T and then samples a random configuration, q_{rand} , from a uniform distribution in the neighborhood of q . Configuration q is selected at random with probability $\pi_T(q)$. The local planner Δ (see section

7.1) attempts a connection between q and q_{rand} . If successful, q_{rand} is added to the vertices of T and (q, q_{rand}) is added to the edges of T . The process is repeated until a specified number of configurations has been added to T . The pseudocode is given in algorithms 8 and 9. Figure 7.12 illustrates this method in the simple case of a point robot in a two-dimensional Euclidean workspace.

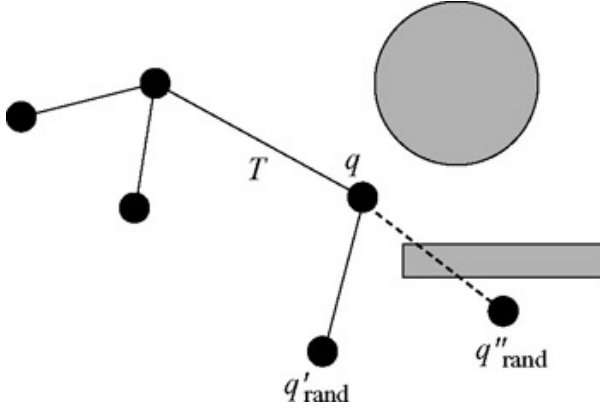


Figure 7.12: Adding a new configuration to an EST. Suppose q is selected and q'_{rand} is created in its neighborhood. The local planner succeeds in connecting q to q'_{rand} . Configuration q'_{rand} and the edge (q, q'_{rand}) are added to the tree T . Had q''_{rand} been created, no nodes or edges would have been added to T , as the local planner would have failed to connect q and q''_{rand} .

Recall that in the roadmap construction of PRM, algorithm 6 in section 7.1, a new random configuration in $\mathcal{Q}_{\text{free}}$ is never rejected but it is immediately added to the roadmap. No attempts are made to connect it to existing configurations in the roadmap. In contrast, in the construction step of EST, a new configuration is added to T only if Δ succeeds in connecting it to an existing configuration in T . It follows then that there is a path from the root of T to every configuration in T .

Algorithm 8: Build EST Algorithm

Input:

q_0 : the configuration where the tree is rooted

n : the number of attempts to expand the tree

Output:

A tree $T = (V, E)$ that is rooted at q_0 and has $\leq n$ configurations

1: $V \leftarrow \{q_0\}$

2: $E \leftarrow \emptyset$

3: **for** $i = 1$ to n **do**

4: $q \leftarrow$ a randomly chosen configuration from T with probability $\pi_T(q)$

5: extend EST (T, q)

6: **end for**

7: **return** T

Algorithm 9: Extend EST Algorithm

Input:

$T = (V, E)$: an EST

q : a configuration from which to grow T

Output:

A new configuration q_{new} in the neighborhood of q , or NIL in case of failure

1: $q_{\text{new}} \leftarrow$ a random collision-free configuration from the neighborhood of q

2: **if** $\Delta(q, q_{\text{new}})$ **then**

3: $V \leftarrow V \cup \{q_{\text{new}}\}$

4: $E \leftarrow E \cup \{(q, q_{\text{new}})\}$

5: **return** q_{new}

```

6: end if
7: return NIL

```

Guiding the Sampling

The effectiveness of **EST** relies on the ability to avoid oversampling any region of $\mathcal{Q}_{\text{free}}$, especially the neighborhoods of q_{init} and q_{goal} . Hence, careful consideration is given to the choice of the probability density function π_T . Ideally, the function π_T should be chosen such that the sampled configurations constitute a rather uniform covering of the connected components of $\mathcal{Q}_{\text{free}}$ containing q_{init} and q_{goal} . A good choice of π is biased toward configurations of T whose neighborhoods are not dense. There are several ways to measure the density of a neighborhood. One that works well in practice associates with each configuration q of T a weight, $w_T(q)$, that counts the number of configurations within some predefined neighborhood of q . If $\pi_T(q)$ is defined to be inversely proportional to $w_T(q)$, then configurations with sparse neighborhoods are more likely to be picked by the planner and used as input to [algorithm 9](#).

The naive method to compute $\pi_T(q)$ enumerates all the configurations of T and tests if they are close to q . This method takes linear time in the number of configurations, n , in the tree T and works well only for relatively small n . A reasonable approximation to $\pi_T(q)$ can be obtained by imposing a grid on \mathcal{Q} . At each iteration, the planner selects the configuration from which to grow the tree by choosing at random a cell and a configuration from this cell. This method was used in [367] and is described in [subsection 7.2.3](#).

Several other π_T functions have been proposed. In [349,350], $\pi_T(q)$ is defined to be a function of the order in which q is generated, its number of neighbors, its out degree, and an A^* cost function A^*_{cost} . The A^*_{cost} is commonly used in graph search to focus the search toward paths with low cost and is computed as the sum of the total cost from the root of the tree to q and the estimated cost from q to the goal configuration. The above weight function combines in a natural way standard **EST** heuristics with potential field methods.

Merging of Trees

The merging of the trees is achieved by pushing the exploration of the space from one tree toward the space explored by the other tree. Initially, a configuration in T_{init} is used as described in [algorithm 9](#) to produce a new configuration q . Then the local planner attempts to connect q to its closest k configurations in T_{goal} . If a connection is successful, the two trees are merged. Otherwise, the trees are swapped and the process is repeated for a specified number of times. [Figure 7.13](#) illustrates the merging of two **EST** trees in a simple case of a two-dimensional Euclidean space.

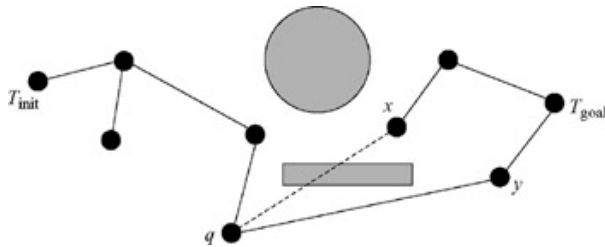


Figure 7.13: Merging two **EST** trees. Configuration q is just added to the first tree, T_{init} . The local planner attempts to connect q to its closest configurations x and y in the second tree, T_{goal} . The local planner fails to connect q to x , but succeeds in the case of y .

The merging of the two trees is obtained by connecting some configuration $q_1 \in T_{\text{init}}$ to some configuration $q_2 \in T_{\text{goal}}$ by using the local planner Δ . Thus, the path between q_{init} and q_{goal} , which are the roots of the corresponding trees, is obtained by concatenating the path from q_{init} to q_1 in T_{init} to the path from q_2 to q_{goal} in T_{goal} .

Care should be taken when implementing **ESTs**. A successful implementation requires a fast update of π_T as new configurations are added to T . The linear cost of the naive method is too high and grid-based approaches or hashing methods (such as those described in [section 7.2.3](#)) must be employed for large n and high-dimensional \mathcal{Q} .

7.2.2 Rapidly-Exploring Random Trees

RRTs were introduced as a single-query planning algorithm that efficiently covers the space between q_{init} and q_{goal} [249,270-272]. The planner was again initially developed for kinodynamic motion planning, where, as in the case of ESTs, a single tree is built. The applicability of RRTs extends beyond kinodynamic planning problems. The RRT algorithm has been shown to be probabilistically complete [271].

Construction of Trees

Let T be one of the trees T_{init} or T_{goal} rooted at q_{init} and q_{goal} , respectively. Each tree T is incrementally extended. At each iteration, a random configuration, q_{rand} , is sampled uniformly in Q_{free} . The nearest configuration, q_{near} , to q_{rand} in T is found and an attempt is made to make progress from q_{near} toward q_{rand} . Usually this entails moving q_{near} a distance `step_size` in the straight line defined by q_{near} and q_{rand} . This newly generated configuration, q_{new} , if it is collision-free, is then added to the vertices of T , and the edge $(q_{\text{near}}, q_{\text{new}})$ is added to the edges of T . The pseudocode is given in algorithms 10 and 11. Figure 7.14 illustrates the extension step of an RRT for a point robot operating in a two-dimensional Euclidean workspace.

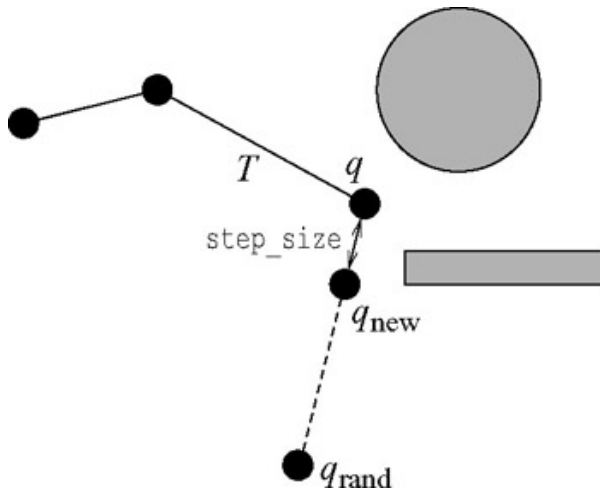


Figure 7.14: Adding a new configuration to an RRT. Configuration q_{rand} is selected randomly from a uniform distribution in Q_{free} . Configuration q is the closest configuration in T to q_{rand} (this configuration is denoted as q_{near} in the algorithm). Configuration q_{new} is obtained by moving q by `step_size` toward q_{rand} . Only q_{new} and the edge (q, q_{new}) are added to the RRT.

The sampling is done by algorithm 11, which produces a new configuration, q_{new} , as a result of moving some configuration q_{near} by `step_size` toward a configuration q_{rand} . A natural question to consider is how `step_size` is determined.

Algorithm 10: Build RRT Algorithm

Input:

q_0 : the configuration where the tree is rooted

n : the number of attempts to expand the tree

Output:

A tree $T = (V, E)$ that is rooted at q_0 and has $\leq n$ configurations

1: $V \leftarrow \{q_0\}$

2: $E \leftarrow \square$

3: **for** $i = 1$ to n **do**

4: $q_{\text{rand}} \leftarrow$ a randomly chosen free configuration

5: extend RRT (T, q_{rand})

```

6: end for
7: return  $T$ 

```

Algorithm 11: Extend RRT Algorithm

Input:
 $T = (V, E)$: an RRT
 q : a configuration toward which the tree T is grown

Output:
A new configuration q_{new} toward q , or NIL in case of failure

```

1:  $q_{\text{near}} \leftarrow$  closest neighbor of  $q$  in  $T$ 
2:  $q_{\text{new}} \leftarrow$  progress  $q_{\text{near}}$  by step_size along the straight line in  $\mathcal{Q}$  between  $q_{\text{near}}$  and  $q_{\text{rand}}$ 
3: if  $q_{\text{new}}$  is collision-free then
4:    $V \leftarrow V \cup \{q_{\text{new}}\}$ 
5:    $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$ 
6: return  $q_{\text{new}}$ 
7: end if
8: return NIL

```

One possible way is to choose `step_size` dynamically based on the distance between q_{near} and q_{rand} as given by the distance function used. It makes sense to choose a large value for `step_size` if the two configurations are far from colliding, and small otherwise. RRT is sensitive to the distance function used, since it is this function that determines `step_size` and guides the sampling. A discussion of the metrics and their effects on RRTs is found in [103]. It is also interesting to consider a greedier alternative that tries to move q_{new} as close to q_{rand} as possible. This method, [algorithm 12](#), calls [algorithm 11](#) until q_{new} reaches q_{rand} or no progress is possible. If [algorithm 12](#) is called in [algorithm 10](#), line 5, an RRT with greater than n nodes may be created.

Algorithm 12: Connect RRT Algorithm

Input:
 $T = (V, E)$: an RRT
 q : a configuration toward which the tree T is grown

Output:
connected if q is connected to T ; failure otherwise

```

1: repeat
2:    $q_{\text{new}} \leftarrow$  extend RRT ( $T, q$ )
3: until ( $q_{\text{new}}=q$  or  $q_{\text{new}}= \text{NIL}$ )
4: if  $q_{\text{new}} = q$  then
5:   return connected
6: else
7:   return failure
8: end if

```

It is important to note the tradeoff that exists between the exploration of \mathcal{Q} and the number of samples added to the tree, especially for high-dimensional problems. If `step_size` is small, then the exploration steps are short and the nodes of the tree are close together. A successful call to [algorithm 12](#) results in many nodes being added to the tree. As the number of nodes becomes large, memory consumption is increased and finding the nearest neighbor becomes expensive, which in turn reduces the performance of the planner. In such cases, it may be better to add only the last sample of the Extend RRT iteration to the tree and no intermediate samples.

Guiding the Sampling

Selecting a node uniformly at random in step 4 of [algorithm 10](#) is a basic mechanism of RRT. It is also of interest to consider other sampling functions that are biased toward the connected components of $\mathcal{Q}_{\text{free}}$ that contain q_{init} or

q_{goal} . Let's consider the case of q_{goal} . At an extreme, a very greedy sampling function can be defined that sets q_{rand} to q_{goal} (if the tree being built is rooted at q_{init}) or to q_{init} (if the tree being built is rooted at q_{goal}). The problem with this approach is that it introduces too much bias, and eventually RRT ends up behaving like a randomized potential field planner that gets stuck in local minima. It seems, therefore, that a suitable choice is a sampling function that alternates, according to some probability distribution, between uniform samples and samples biased toward regions that contain the initial or the goal configuration. Experimental evidence [249,271] has shown that setting q_{rand} to q_{goal} with probability p , or randomly generating q_{rand} with probability $1 - p$ from a uniform distribution, works well. Even for small values of p , such as 0.05, the tree rooted at q_{init} converges much faster to q_{goal} than when just uniform sampling is used. This simple function can be further improved by sampling in a region around q_{goal} instead of setting q_{rand} to q_{goal} . The region around q_{goal} is defined by the vertices of the RRT closest to q_{goal} at each iteration of the construction step.

Merging of Trees

In the merging step, RRT tries to connect the two trees, T_{init} and T_{goal} , rooted at q_{init} and q_{goal} , respectively. This is achieved by growing the trees toward each other. Initially, a random configuration, q_{rand} , is generated. RRT extends one tree toward q_{rand} and as a result obtains a new configuration, q_{new} . Then the planner attempts to extend the closest node to q_{new} in the other tree toward q_{new} . If successful, the planner terminates, otherwise the two trees are swapped and the process is repeated a certain number of times. Figure 7.15 illustrates a simple case of merging two RRTs in a two-dimensional Euclidean space.

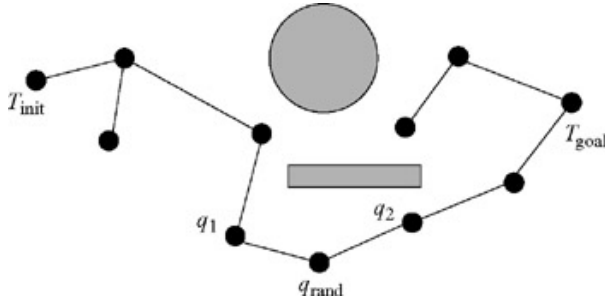


Figure 7.15: Merging two RRTs. Configuration q_{rand} is generated randomly from a uniform distribution in Q_{free} . Configuration q_1 was extended to q_{rand} . q_2 is the closest configuration to q_{rand} in T_{goal} . It was possible to extend q_2 to q_{rand} . As a result, T_{init} and T_{goal} were merged.

The merge algorithm, as presented in algorithm 13, uses algorithm 11. Recall that algorithm 11 produces a new configuration that is only `step_size` away from the nearest node in the existing RRT. By replacing algorithm 11 in lines (3) and (5) with algorithm 12, new configurations are produced farther away. This greedier approach has been reported to work well [249, 271]. It is also reasonable to replace only one of the algorithm 11 calls, and thus obtain a balance between the two approaches. The choice of whether a greedier or more balanced approach is used for the exploration depends on the particular problem being solved. Discussions can be found in [249, 271]. Once the two RRTs are merged together, a path from q_{init} to q_{goal} is obtained in the same way as in the case of ESTs.

Algorithm 13: Merge RRT Algorithm

Input:

T_1 : first RRT

T_2 : second RRT

l : number of attempts allowed to merge T_1 and T_2

Output:

merged if the two RRTs are connected to each other; failure otherwise

```

1: for  $i = 1$  to  $l$  do
2:    $q_{rand} \leftarrow$  a randomly chosen free configuration
3:    $q_{new, 1} \leftarrow$  extend RRT ( $T_1$ ,  $q_{rand}$ )
4:   if  $q_{new, 1} \neq \text{NIL}$  then
5:      $q_{new, 2} \leftarrow$  extend RRT ( $T_2$ ,  $q_{new, 1}$ )
6:     if  $q_{new, 1} = q_{new, 2}$  then
```

```

7:         return merged
8:     end if
9:     SWAP( $T_1$ ,  $T_2$ )
10: end if
11: end for
12: return failure

```

The implementation of **RRT** is easier than that of **EST**. Unlike **EST**, **RRT** does not compute the number of configurations lying inside a predefined neighborhood of a node and it does not maintain a probability distribution for its configurations.

7.2.3 Connection Strategies and the SBL Planner

Lazy evaluation, which was introduced as a connection strategy in [section 7.1.4](#), can also be used in the context of tree-building sampling methods. A combination of lazy evaluation and **ESTs** has been presented in the context of the Single-query, Bi-directional, Lazy collision-checking (**SBL**) planner [367].

SBL constructs two **EST** trees rooted at q_{init} and q_{goal} . **SBL** creates new samples according to the **EST** criteria but does not immediately test connections between samples for collisions. A connection between two configurations is checked exactly once and this is done only when the connection is part of the path joining the two trees together (if such a path is found). This results in substantial time-savings as reported in [367].

It is also worth noting that **SBL** uses a clever way to find which configurations to expand and hence guide the sampling (see [section 7.2.1](#)). In the original **EST**, a configuration is chosen for expansion according to the density of sampling in its neighborhood. Finding neighbors is in general an expensive operation as the dimension increases. **SBL** imposes a coarse grid on \mathcal{Q} . It then picks randomly a non-empty cell in the grid and a sample from that cell. The probability to pick a certain sample is greater if this sample lies in a cell with few nodes. This simple technique allows a fast implementation of **SBL** and is applicable to all **EST**-based planners. Details on how this technique helps in the connection of trees grown from the initial and goal configurations are given in [367].

EST and **RRT** employ excellent sampling and connection schemes that can be further exploited to obtain even more powerful planners, as discussed in the [next section](#).

 [Previous](#)

[Next](#) 



Chapter 7 - Sampling-Based Algorithms

Principles of Robot Motion: Theory, Algorithms, and Implementation

by Howie Choset et al.

The MIT Press © 2005

◀ Previous

Next ▶

7.3 Integration of Planners: Sampling-Based Roadmap of Trees

This section shows how to effectively combine a sampling-based method primarily designed for multiple-query planning (PRM) with sampling-based tree methods primarily designed for single-query planning (EST, RRT, and others). The Sampling-Based Roadmap of Trees (SRT) planner [14, 43, 353] takes advantage of the local sampling schemes of tree planners to populate a PRM-like roadmap. SRT replaces the local planner of PRM with a single-query sampling-based tree planner enabling it to solve problems that other planners cannot.

A question arises as to whether SRT is a multiple-query or single-query planner. SRT can be seen as a multiple-query planner, since once the roadmap is constructed, SRT can use the roadmap to answer multiple queries. SRT can also be seen as a singlequery planner because for certain very difficult problems, the cost of constructing a roadmap and solving a query by SRT is less than that of any single-query planner solving the same query. This is why in [section 7.1](#) it was pointed out that the distinction of planners to multiple-query and single-query planners is very useful for describing the planners, but always needs to be placed in perspective given the planning problem at hand.

As in the PRM formulation, SRT constructs a roadmap aiming at capturing the connectivity of Q_{free} . The nodes of the roadmap are not single configurations but trees, as illustrated in [figure 7.16](#). Connections between trees are computed by a bidirectional tree algorithm such as EST or RRT. Recall that a roadmap is an undirected graph $G = (V, E)$ over a finite set of configurations $V \subset Q_{\text{free}}$, and each edge $(q', q'') \in E$ represents a local path from q' to q'' . SRT constructs a roadmap of trees. The undirected graph $G_T = (V_T, E_T)$ is an induced subgraph of G which is defined by partitioning G into a set of subgraphs T_1, \dots, T_n , which are trees, and contracting them into the vertices of G_T . In other words, $V_T = \{T_1, \dots, T_n\}$ and $(T_i, T_j) \in E_T$ if there exist configurations $q_i \in T_i$ and $q_j \in T_j$ such that q_i and q_j have been connected by a local path.

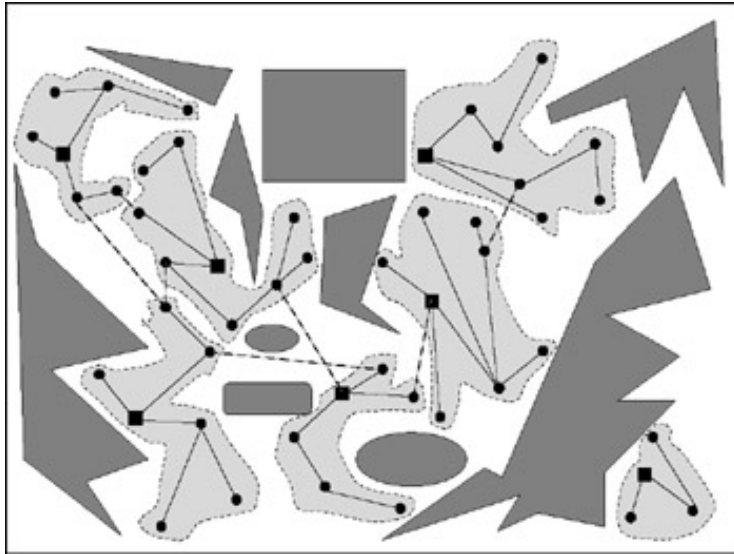


Figure 7.16: An example of a roadmap for a point robot in a two-dimensional workspace. The dark gray areas are obstacles. Each node of the roadmap is a tree rooted at the black squares. The thin-solid lines indicate connections between configurations of the same tree. The thick-dashed lines indicate connections between configurations of two different trees. The light gray areas delineate the separate trees.

Adding Trees to the Roadmap

In **SRT**, the trees of the roadmap G_T are computed by sampling their roots uniformly at random in Q_{free} , and then growing the trees using a sampling-based tree planner, such as algorithms 8 and 10. Note that in principle any of the node-sampling strategies of **PRM** described in [section 7.1.3](#) can be applied.

Adding Edges to the Roadmap

The roadmap construction is not yet complete since no edges have been computed. An edge between two trees indicates that they are merged into one. For each tree T_i , a set N_{T_i} consisting of closest and random tree neighbors is computed and a connection is attempted between T_i and each tree T_j in N_{T_i} . As in **PRM**, **SRT** may choose to avoid the computation of candidate edges that cannot decrease the number of connected components in G_T . In fact, any of the **PRM** connection strategies of [section 7.1.4](#) can be applied here. In order to determine the closest neighbors, each tree T_i defines a representative configuration q_{T_i} which is computed as an aggregate of the configurations in T_i . The distance between two trees T_i and T_j is defined as $\text{dist}(q_{T_i}, q_{T_j})$. It has been observed experimentally in [14,43] that the consideration of random neighbors offsets some of the problems introduced by the distance function used.

Computation of candidate edges is typically carried out by a sampling-based tree planner. First, for each candidate edge (T_i, T_j) , a number of close pairs of configurations of T_i and T_j are quickly checked with a fast deterministic local planner. If a local path is found, no

further computation takes place. Otherwise, the sampling-based tree planner used to add trees to the roadmap should be employed. During tree connection, additional configurations are typically added to the trees T_i and T_j . If the connection is successful, the edge (T_i, T_j) is added to E_T and the graph components to which T_i and T_j belonged are merged into one. Note that the trees T_i and T_j are connected when some configuration $q_i \in T_i$ is connected to some configuration $q_j \in T_j$.

The pseudocode is given in [algorithm 14](#). In addition to `RRT` and `EST`, other sampling-based tree planners, such as [251, 350], can be used with `SRT`. It is also possible to incorporate lazy evaluation into `SRT` by using a planner similar to `SBL` for tree expansion and edge computations.

Algorithm 14: Connect `SRT` Algorithm

Input:

V_T : a set of trees

k : number of closest neighbors to examine for each tree

r : number of random neighbors to examine for each tree

Output:

A roadmap $G_T = (V_T, E_T)$ of trees

```

1:  $E_T \leftarrow \square$ 
2: for all  $T_i \in V_T$  do
3:    $N_{T_i} \leftarrow k$  nearest and  $r$  random neighbors of  $T_i$  in  $V_T$ 
4:   for all  $T_j \in N_{T_i}$  do
5:     if  $T_i$  and  $T_j$  are not in the same connected component of  $G_T$  then
6:       merged  $\leftarrow$  FALSE
7:        $S_i \leftarrow$  a set of randomly chosen configurations from  $T_i$ 
8:       for all  $q_i \in S_i$  and merged = FALSE do
9:          $q_j \leftarrow$  closest configuration in  $T_j$  to  $q_i$ 
10:        if  $\Delta(q_i, q_j)$  then
11:           $E_T \leftarrow E_T \cup \{(T_i, T_j)\}$ 
12:          merged  $\leftarrow$  TRUE
13:        end if
14:      end for
15:      if merged = FALSE and Merge Trees ( $T_i, T_j$ ) then
16:         $E_T \leftarrow E_T \cup \{(T_i, T_j)\}$ 
17:      end if
18:    end if
19:  end for
20: end for

```

Answering Queries

As in `PRM`, the construction of the roadmap enables `SRT` to answer multiple queries

efficiently if needed. Given q_{init} and q_{goal} , the trees T_{init} and T_{goal} rooted at q_{init} and q_{goal} , respectively, are grown for a small number of iterations and added to the roadmap. Neighbors of T_{init} and T_{goal} are computed as a union of the k closest and r random trees, as described previously. The tree-connection algorithm alternates between attempts to connect T_{init} and T_{goal} to each of their respective neighbor trees. A path is found if at any point T_{init} and T_{goal} lie in the same connected component of the roadmap. In order to determine the sequence of configurations that define a path from q_{init} to q_{goal} , it is necessary to find the sequence of trees that define a path from T_{init} to T_{goal} and then concatenate the local paths between any two consecutive trees. Path smoothing can be applied to the resulting path to improve the quality of the output.

Parameters of SRT

A nice feature of SRT is that it can behave exactly as PRM, RRT, or EST. That is, if the number of configurations in a tree is one, the number of close pairs is one and the number of iterations to run the bi-directional tree planner is zero (denoted by Merge Trees in line (15) of [algorithm 14](#)), then SRT behaves as PRM. If the number of trees in the roadmap is zero and the number of close pairs is zero, then SRT behaves as RRT or EST depending on the type of tree. SRT provides a framework where successful sampling schemes can be efficiently combined.

Parallel SRT

SRT is significantly more decoupled than tree planners such as ESTs and RRTs. Unlike ESTs and RRTs, where the generation of one configuration depends on all previously generated configurations, the trees of SRT can be generated independently of one another. This decoupling allows for an efficient parallelization of SRT [14]. By increasing the power of the local planner and by using trees as nodes of the roadmap, SRT distributes its computation evenly among processors, requires little communication, and can be used to solve very high-dimensional problems and problems that exceed the resources available to the sequential implementation [14]. Adding trees to the roadmap can be parallelized efficiently, since there are no dependencies between the different trees. Adding edges to the roadmap is harder to parallelize efficiently. Since trees can change after an edge computation and since computing an edge requires direct knowledge of both trees, the edge computations cannot be efficiently parallelized without some effort [14]. Furthermore, if any computation pruning according to the sparse roadmaps heuristic is done (see [section 7.1.4](#)), this will entail control flow dependencies throughout the computation of the edges.

 [Previous](#)

[Next](#) 



Chapter 7 - Sampling-Based Algorithms

Principles of Robot Motion: Theory, Algorithms, and Implementation

by Howie Choset et al.

The MIT Press © 2005

◀ Previous

Next ▶

7.4 Analysis of PRM

The planners discussed in this chapter sample points in Q_{free} and connect them using a local planner. As opposed to exact motion-planning algorithms, such as [90, 306, 361, 373, 375], it is possible that PRM and other sampling-based motion planners can report falsely that no path exists. It would seem that the correctness of the motion planner has been sacrificed in favor of good experimental performance. This, however, is not exactly the case. Rather than being a purely heuristic technique, a weaker completeness property, called probabilistic completeness, can be proved to hold for PRM as was discussed in the introductory section of this chapter.

This section deals with probabilistic completeness proofs and analyses of the basic PRM planner. In the basic PRM planner, samples are chosen from a uniform random distribution. Although the presented results are for an idealized version of PRM, it is strongly conjectured that probabilistic completeness results can be extended to conditional random sampling and to deterministic sampling, in the latter case, in the form of resolution completeness results.

Suppose that $a, b \in Q_{\text{free}}$ can be connected by a path in Q_{free} . PRM is considered to be probabilistically complete, if for any given (a, b)

$$\lim_{n \rightarrow \infty} \Pr[(a, b) \text{ FAILURE}] = 0,$$

where $\Pr[(a, b) \text{ FAILURE}]$ denotes the probability that PRM fails to answer the query (a, b) after a roadmap with n samples has been constructed. The number of samples gives a measure of the work that needs to be done and hence it can be used as a measure of the complexity of the algorithm.

The results presented in this section apply to the basic PRM algorithm. [Section 7.4.1](#) analyzes the operation of PRM in a Euclidean space. Using this analysis, it is possible to gain an estimate on how much work (as measured by the number of generated samples) is needed to produce paths with certain properties. [Section 7.4.2](#) shows how certain goodness properties of the underlying space affect the performance of PRM. It is this analysis that sheds light on why PRM works well with extremely simple sampling strategies such as uniform sampling. Experimental observations indicate that many of the path-planning problems that arise in physical settings have goodness properties, such as the ones described in [section 7.4.2](#), that may not require elaborate sampling schemes. Both

analyses prove probabilistic completeness for PRM. Section 7.4.3 shows an equivalence between the probabilistic completeness of PRM and a much simpler planner.

7.4.1 PRM Operating in \mathbb{R}^d

This section provides an analysis [222,223] of PRM operating in Euclidean \mathbb{R}^d . Assuming that a path between two different configurations a and b exists, it is shown that the probability of PRM failing to connect a and b depends on (1) the length of the known path, (2) the distance of the path from the obstacles, and (3) the number of configurations in the roadmap. Connecting a and b by a long path requires a larger number of intermediate configurations to be present in the roadmap. Paths that are closer to obstacles are harder to obtain because of potential collisions. Similarly, paths that are inside narrow passages are harder to obtain because the probability of placing random configurations inside narrow passages is small. The probabilistic completeness of PRM is proved by tiling the known path with a set of carefully chosen balls and showing that generating a point in each ball ensures that a path between a and b will be found.

Let Q_{free} be an open subset of $[0, 1]^d$ and let dist be the Euclidean metric on \mathbb{R}^d . The local planner of PRM connects points $a, b \in Q_{\text{free}}$ when the straight-line ab lies in Q_{free} . A path γ in Q_{free} from a to b consists of a continuous map $\gamma : [0, 1] \rightarrow Q_{\text{free}}$, where $\gamma(0) = a$ and $\gamma(1) = b$. The clearance of a path, denoted $\text{clr}(\gamma)$, is the farthest distance away from the path at which a given point can be guaranteed to be in Q_{free} . If a path γ lies in Q_{free} , then $\text{clr}(\gamma) > 0$.

The measure μ denotes the volume of a region of space, e.g., $\mu([0, 1]^d) = 1$. For any measurable subset $A \subset \mathbb{R}^d$, $\mu(A)$ is its volume. For example, an open ball of radius ϵ centered at x is denoted by $B_\epsilon(x)$ and its volume is given by $\mu(B_\epsilon(x))$. The uniform distribution is used by PRM to sample points. If $A \subset Q_{\text{free}}$ is a measurable subset and x is a random point chosen from Q_{free} by the point-sampling function of PRM, then

$$\Pr(x \in A) = \frac{\mu(A)}{\mu(Q_{\text{free}})}.$$

THEOREM 7.4.1

Let $a, b \in Q_{\text{free}}$ such that there exists a path γ between a and b lying in Q_{free} . Then the probability that PRM correctly answers the query generating n configurations is given by

$$\Pr[(a, b) \text{ SUCCESS}] = 1 - \Pr[(a, b) \text{ FAILURE}] \geq 1 - \left\lceil \frac{2L}{\rho} \right\rceil e^{-\sigma \rho^d n},$$

where L is the length of the path γ , $\rho = \text{clr}(\gamma)$, $B_1(\cdot)$ is the unit ball in \mathbb{R}^d and

$$\sigma = \frac{\mu(B_1(\cdot))}{2^d \mu(Q_{\text{free}})}.$$

Proof

Let $\rho = \text{clr}(\gamma)$ and note that $\rho > 0$. Let $m = \left\lceil \frac{2L}{\rho} \right\rceil$ and observe that there are m points on the path $a = x_1, \dots, x_m = b$ such that $\text{dist}(x_i, x_{i+1}) < \rho/2$. Let $y_i \in B_{\rho/2}(x_i)$ and $y_{i+1} \in B_{\rho/2}(x_{i+1})$. Then the line segment $y_i y_{i+1}$ must lie inside Q_{free} since both endpoints lie in the ball $B_\rho(x_i)$. An illustration of this basic fact is given in figure 7.17. Let $V \subset Q_{\text{free}}$ be a set of n configurations generated uniformly at random by PRM . If there is a subset of configurations $\{y_1, \dots, y_m\} \subset V$ such that $y_i \in B_{\rho/2}(x_i)$, then a path from a to b will be contained in the roadmap. Let I_1, \dots, I_m be a set of indicator variables such that each I_i witnesses the event that there is a $y \in V$ and $y \in B_{\rho/2}(x_i)$. It follows that PRM succeeds in answering the query (a, b) if $I_i = 1$ for all $1 \leq i \leq m$. Therefore,

$$\Pr[(a, b) \text{ FAILURE}] \leq \Pr \left(\bigvee_{i=1}^m I_i = 0 \right) \leq \sum_{i=1}^m \Pr[I_i = 0],$$

where the last inequality follows from the union bound [119].

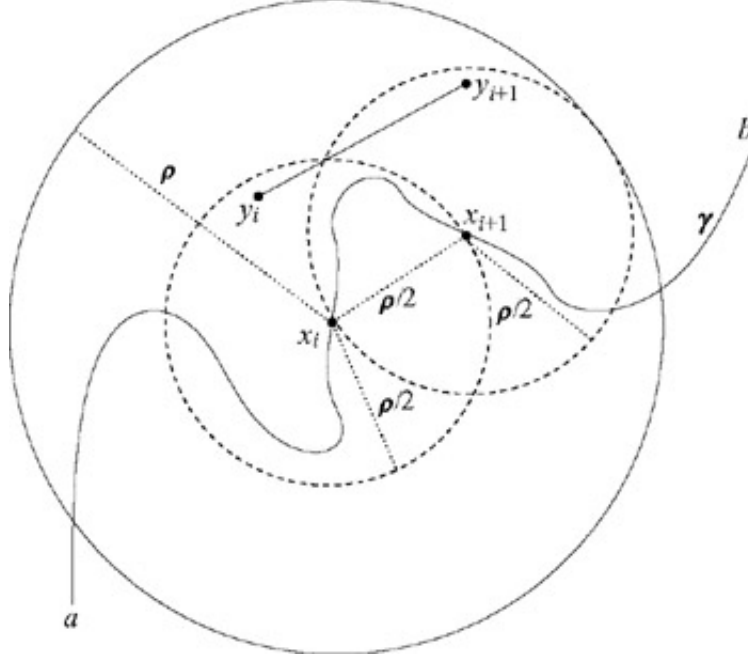


Figure 7.17: Points y_i and y_{i+1} are inside the $\rho/2$ balls and straight-line $y_i y_{i+1}$ is in Q_{free} .

The events $I_i = 0$ are independent since the samples are independent. The

probability of a given $I_i = 0$ is computed by observing that the probability of a single randomly generated point falling in $B_{\rho/2}(x_i)$ is $\mu(B_{\rho/2}(x_i))/\mu(Q_{\text{free}})$. It follows that the probability that none of the n uniform, independent samples falls in $B_{\rho/2}(x_i)$ satisfies

$$Pr[I_i = 0] = \left(1 - \frac{\mu(B_{\rho/2}(x_i))}{\mu(Q_{\text{free}})}\right)^n.$$

Since the sampling is uniform and independent, then

$$Pr[(a, b) \text{ FAILURE}] \leq \left\lceil \frac{2L}{\rho} \right\rceil \left(1 - \frac{\mu(B_{\rho/2}(\cdot))}{\mu(Q_{\text{free}})}\right)^n.$$

However

$$\frac{\mu(B_{\rho/2}(\cdot))}{\mu(Q_{\text{free}})} = \frac{\left(\frac{\rho}{2}\right)^d \mu(B_1(\cdot))}{\mu(Q_{\text{free}})} = \sigma \rho^d,$$

for σ defined as in the statement of this theorem. The bound is obtained by using the relation $(1 - \beta)^n \leq e^{-\beta n}$ for $0 \leq \beta \leq 1$:

$$Pr[(a, b) \text{ FAILURE}] \leq \left\lceil \frac{2L}{\rho} \right\rceil e^{-\sigma \rho^d n}.$$

As shown from the proof above, a better estimate for $Pr[(a, b) \text{ FAILURE}]$ is available than the exponential bound given in [theorem 7.4.1](#). The exponential bound is a simplification that allows the direct calculation of n when the user wishes to specify an acceptable value for $Pr[(a, b) \text{ FAILURE}]$. The proof of [theorem 7.4.1](#) can be extended to take into account that clearance can vary along the path [223]. [Theorem 7.4.1](#) implies that PRM is probabilistically complete. Moreover, the probability of failure converges exponentially quickly to 0.

7.4.2 (ϵ , α , β)-Expansiveness

This section argues how PRM roadmaps capture the connectivity of Q_{free} based on the analysis of [192, 196, 197, 228, 229]. A principal intuition behind PRM has been that in spaces that are not "pathologically" difficult, that is in spaces where reasonable assumptions about connectivity hold, the planner will do well even with simple sampling schemes such as random sampling.

Observe that, in the general case, Q_{free} can be broken into a union of disjoint connected components $\{Q_{\text{free}1}, \dots, Q_{\text{free}i}, \dots\}$. Let $G = (V, E)$ be the roadmap

constructed by PRM with uniform sampling. For each Q_{free_i} , let $V_i = V \cap Q_{free_i}$ and let G_i be the subgraph of G induced by V_i . In the rest of this section, it is shown how to determine the number of configurations that should be generated to ensure that, with probability exceeding a given constant, each G_i is connected.

Given a subset S of Q_{free} , the reachable set from S is the set of configurations in Q_{free} that are visible from any configuration in S . Figures 7.18(a) and (b) show an example.

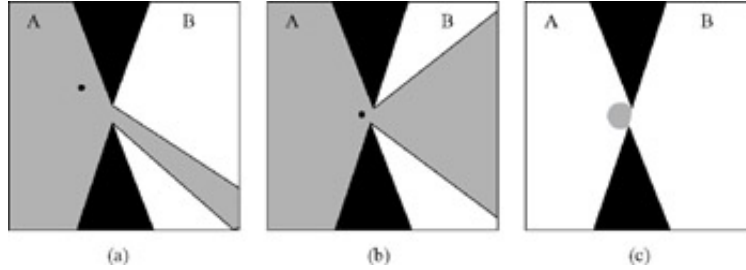


Figure 7.18: The areas in black indicate the obstacles. In (a) and (b), the areas in gray indicate the reachability sets of the two points represented by black circles. In (c), the set A has a small lookout (the gray area), because only a small subset of points in A near the narrow passage can see a large fraction of points in B . (From Hsu [192].)

DEFINITION 7.4.1

Let $S \subset Q_{free}$. The reachable set of S is defined as

$$\text{reach}(S) = \{x \in Q_{free} \mid \exists y \in S \text{ such that } \overline{xy} \subset Q_{free}\}.$$

The shorthand $\text{reach}(x)$ is used instead of $\text{reach}(\{x\})$ when $x \in Q_{free}$.

A space Q_{free} is ϵ -good if the volume of Q_{free} that each point in Q_{free} can reach is at least an ϵ fraction of the total free volume of Q_{free} .

DEFINITION 7.4.2

Let ϵ be a constant in $(0, 1]$. A space Q_{free} is ϵ -good if for all $x \in Q_{free}$,

$$\mu(\text{reach}(x)) \geq \epsilon \mu(Q_{free}).$$

The β -lookout of a subset S of a connected component of Q_{free_i} is the subset of S for which each configuration in that subset can reach more than a β fraction of $Q_{free_i} \setminus S$. An example is given in figure 7.18(c).

DEFINITION 7.4.3

Let β be a constant in $(0, 1]$ and let S be a subset of a connected component Q_{free} of Q_{free} . The β -lookout set of S is defined as

$$\text{lookout}_{\beta}(S) = \{x \in S \mid \mu(\text{reach}(x) \setminus S) \geq \beta \mu(Q_{\text{free}} \setminus S)\}.$$

The following definition captures how reachability spreads across the space.

DEFINITION 7.4.4

Let ϵ , α and β be constants in $(0, 1]$. A space is $(\epsilon, \alpha, \beta)$ -expansive if

1. it is ϵ -good, and
2. for any connected subset of $S \subset Q_{\text{free}}$, $\mu(\text{lookout}_{\beta}(S)) \geq \alpha \mu(S)$.

The first condition of [definition 7.4.4](#) ensures that a certain fraction of Q_{free} is visible from any configuration in Q_{free} . The second condition ensures that each subset $S \subseteq Q_{\text{free}}$ has a large lookout set. It is reasonable to think of S as the union of the reachability sets of a set V of points. Large values of α and β indicate that it is easy to choose random points from S such that adding them to V results in significant expansion of S . This is desirable since it allows for a quick exploration of the entire space. [Figure 7.19](#) gives an example of an expansive space and indicates the values of ϵ , α and β .

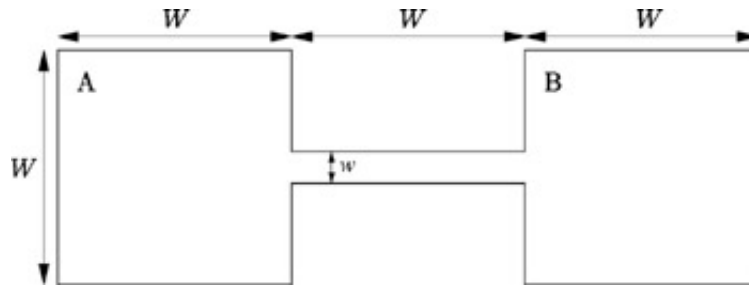


Figure 7.19: An example of an $(\epsilon, \alpha, \beta)$ -expansive Q_{free} with $\epsilon, \alpha, \beta \approx w/W$. The points with the smallest ϵ are located in the narrow passage between square A and square B. Each such point sees only a subset of Q_{free} of volume approximately $3wW$. Hence $\epsilon \approx w/W$. A point near the top right corner of square A sees the entire square; but only a subset of A, of approximate volume wW , contains points that each see a set of volume $2wW$; hence $\alpha \approx w/W$ and $\beta \approx w/W$. (From Hsu [192].)

We now introduce the concept of a linking sequence, which will be used in the development that follows.

DEFINITION 7.4.5

A linking sequence of length ℓ for a configuration $x \in \mathcal{Q}_{\text{free}}$ is a set of configurations $x_1 = x, x_2, \dots, x_\ell$ with an associated sequence of reachable sets $X_1 = \text{reach}(x_1), X_2, \dots, X_\ell \subset \mathcal{Q}_{\text{free}}$, where for all $1 < i \leq \ell$,

$$x_i \in \text{lookout}_\beta(X_{i-1}) \quad \text{and} \quad X_i = X_{i-1} \cup \text{reach}(x_i).$$

The proof of the main result relies on two technical lemmas, whose proofs are given in [192]. **Lemma 7.4.6** gives a bound on the probability of sampling a linking sequence for a given configuration x in terms of α, ε , and t , the length of the linking sequence.

LEMMA 7.4.6

Let V be a set of n configurations chosen independently and uniformly at random from $\mathcal{Q}_{\text{free}}$. Let $s = 1/\alpha\varepsilon$. Given any configuration $x \in V$, there exists a linking sequence in V of length t for x with probability at least $1 - se^{-(n-t-1)/1}$

Lemma 7.4.7 gives a lower bound on the volume of V_t for an arbitrary linking sequence of length t .

LEMMA 7.4.7

Let $x_1 = x, x_2, \dots, x_t$ be a length t linking sequence for $x \in \mathcal{Q}_{\text{free}i}$, where $\mathcal{Q}_{\text{free}i}$ is a connected component of $\mathcal{Q}_{\text{free}}$. Let X_1, X_2, \dots, X_t be the associated reachable sets. If $t \geq \beta^{-1} \ln(4)$, then

$$\mu(X_t) \geq \frac{3\mu(\mathcal{Q}_{\text{free}i})}{4}.$$

The main result of this section follows. Given a number δ , the theorem finds n such that if $2n + 2$ configurations are sampled, then each subgraph G_i is a connected graph with probability at least $1 - \delta$. This indicates that the connectivity of the roadmap G conforms to the connectivity of $\mathcal{Q}_{\text{free}}$. It means that, with high probability, no two connected components of G lie in the same connected component of $\mathcal{Q}_{\text{free}}$.

THEOREM 7.4.2

Let δ be a constant in $(0, 1]$. Suppose a set V of $2n+2$ configurations for

$$n = \left\lceil \frac{8 \ln \left(\frac{8}{\epsilon \alpha \delta} \right)}{\epsilon \alpha} + \frac{3}{\beta} \right\rceil,$$

is chosen independently and uniformly at random from $\mathcal{Q}_{\text{free}}$. Then, with probability at least $1 - \delta$, each subgraph G_i is a connected graph.

Proof

Let x and y be any two configurations in the same connected component $\mathcal{Q}_{\text{free},i}$. Divide the remaining configurations into two sets V' and V'' of n configurations each. By lemma 7.4.6, there is a linking sequence of length t for x in V' with probability at least $1 - se^{-(n-t)/s}$. The same holds true for y and V . Let $X_t(x)$ and $X_t(y)$ be the reachability sets determined by the linking sequences of length t of x and y . By choosing $t \geq 1.5\beta$, lemma 7.4.7 is applied to ensure that $\mu(X_t(x))$ and $\mu(X_t(y))$ are larger than $3\mu(\mathcal{Q}_{\text{free},i})/4$. It follows that $\mu(X_t(x) \cap X_t(y)) \geq \mu(\mathcal{Q}_{\text{free},i})/2$. It is known that $\mu(\mathcal{Q}_{\text{free},i}) \geq \epsilon$, because $\mathcal{Q}_{\text{free},i}$ is an ϵ -good space; the visibility region of any point in $\mathcal{Q}_{\text{free},i}$ must have volume at least ϵ . Since the configurations in V'' are sampled independently and uniformly at random, it follows that with probability at least $1 - (1 - \epsilon/2)^n \geq 1 - e^{-n\epsilon/2}$, there is a configuration in V'' that lies in the intersection of the reachability sets. This means that there is a path from x to y in G_i .

Let B be the event that x and y fail to connect in G_i . By applying a union bound and by the linking sequence construction, it follows that

$$\Pr[B] \leq 2se^{-(n-t)/s} + e^{-n\epsilon/2}.$$

By choosing $n \geq 2t$ and recalling that $s = 1/\alpha\epsilon$,

$$\Pr[B] \leq 2se^{-n/2s} + e^{-n\epsilon/2} \leq 2se^{-n/2s} + e^{-n/2\alpha\epsilon} \leq 3se^{-n/2s}.$$

A graph G_i will fail to be connected if some pair $x, y \in V_i$ fails to be connected.

There are at most $\binom{n}{2}$ such pairs and the probability of this occurring is at most

$$\binom{n}{2} \Pr[B] \leq \binom{n}{2} 3se^{-n/2s} \leq 2n^2 se^{-n/2s} \leq 2se^{(-n-4s \ln n)/2s} \leq 2se^{-n/4s},$$

where the last inequality follows from the observation that $n/2 \geq 4s \ln n$ for $n \geq 8s \ln(8s)$. By requiring that $n \geq 8s \ln(8s/\delta)$, it follows that

$$2se^{-n/4s} \leq 2se^{-2 \ln(8s/\delta)} \leq 2s \left(\frac{\delta^2}{8s} \right) \leq \delta.$$

It is sufficient to choose $n \geq 8s \ln(8s/\delta) + 2t$ for this argument to succeed. By substituting $s = 1/\alpha\epsilon$ and $t = 1.5\beta$ into this expression, the stated result is obtained.

Theorem 7.4.2 implies probabilistic completeness, although some additional argumentation is needed. The main limitation of the above analysis is the reliance on the α , β , and ϵ constants being nonzero. This will be true for any polyhedral space. Since any configuration space can be well approximated with a polyhedral space without changing its connectivity, **theorem 7.4.2** holds. A detailed analysis can be found in [37,192,196,197].

7.4.3 Abstract Path Tiling

In this section, **theorem 7.4.1** is generalized by reducing the set of assumptions to a bare minimum. The new assumptions are sufficient for defining the planner's sampling scheme and the notion of reachability. In fact, the structural requirements for the configuration space are very simple and are captured by the mathematical abstraction of a probability space: essentially a space over which probability can be defined. In the new framework, the balls used to tile a path in **theorem 7.4.1** can be replaced with arbitrary sets of strictly positive measure. These sets are not necessarily connected or open. The analysis is introduced in order to consider **PRM** operating on motion-planning problems with difficult configuration spaces, and with complex local planners such as those arising from motion planning with dynamics, deformable objects, objects with contact, and others [252]. The framework presented in this section enables a rigorous treatment of asymmetric reachability, nonmanifold configuration spaces, and sampling from arbitrary distributions. Hence, it reveals the applicability of the **PRM** scheme to problems beyond basic path-planning. A detailed analysis can be found in [252].

As before, the distribution for configuration generation is encoded with the probability measure μ . So if $A \subset Q_{\text{free}}$, then $\mu(A)$ is the probability that a random sample from Q_{free} lies in A . The local path planner is further generalized away from a straightline planner and is instead replaced with an arbitrary binary relation, R . Informally, xRy means y can be reached using the local planner from x . Note that xRy need not imply yRx . More precisely, the set $R \subset Q_{\text{free}} \times Q_{\text{free}}$ is the set of all query configurations that can be connected by the local planner. For example, if $Q_{\text{free}} \subset [0, 1]^d$ and the local planner is a straight-line planner, then

$$(x, y) \in R \Leftrightarrow \overline{xy} \subset Q_{\text{free}}.$$

It is required that R is measurable in $Q_{\text{free}} \times Q_{\text{free}}$. Membership in R is written interchangeably as $(x, y) \in R$ or xRy .

Algorithm 15: Random Incremental Algorithm

```

1:  $x_0 \leftarrow x$ 
2:  $\ell \leftarrow 0$ 
3: loop
```

```

4:   Check if  $x_\ell R y$ , if so return  $x_0, \dots, x_\ell, y$  as the computed path
5:   Generate  $x_{\ell+1}$  at random from distribution  $\mu$ 
6:   Check if  $x_\ell R x_{\ell+1}$ , if not return no path
7:  $\ell \leftarrow \ell + 1$ 
8: end loop

```

This section develops two distinct ideas from these definitions. A simple motion planner based on random incremental construction of a path is stated in [algorithm 15](#). First, it will be shown that PRM is probabilistically complete if and only if [algorithm 15](#) can answer correctly on every query with nonzero probability. Second, it is proved that probabilistic completeness implies a bound on $Pr[(x, y) \text{ FAILURE}]$ similar to the one stated in [theorem 7.4.1](#).

THEOREM 7.4.3

[Algorithm 15](#) succeeds with nonzero probability on every query if and only if PRM is probabilistically complete. Furthermore, if PRM is probabilistically complete, then there exist constants $\ell \geq 0$ and $p > 0$ such that

$$Pr[(x, y) \text{ FAILURE}] \leq \ell e^{-pn},$$

where n is the number of configurations in the roadmap.

Proof First, the equivalence between PRM and [algorithm 15](#) is proven. Suppose that [algorithm 15](#) succeeds on query (x, y) with probability $P > 0$. The probability of generating each intermediate point along the path from x to y is the same for [algorithm 15](#) and PRM, since they both sample randomly from the same distribution μ . Hence, PRM succeeds on query (x, y) with probability $P > 0$.

For the converse, suppose that after constructing the roadmap, PRM succeeds on query (x, y) with probability $P > 0$. Choose n to be the minimum number of configurations in the roadmap for the previous statement to be true. Since n is the smallest such number, then every configuration of the roadmap appears exactly once as an intermediate point of the path connecting x to y . Note that it does not matter in what order the configurations are generated: the roadmap is permutation invariant. Since the samples are independent, it suffices to consider only the solutions where the path is generated in order and conclude that the

probability of this occurring is then $\frac{P}{n!} > 0$. [Algorithm 15](#), after running for n iterations, would have probability at least $\frac{P}{n!} > 0$ of succeeding.

This concludes the proof of the probabilistic completeness equivalence between [algorithm 15](#) and PRM. It remains to show that the probability of failure for PRM decreases exponentially with the number of samples generated.

Define R^ℓ to be the ℓ th iteration of R , i.e.,

$$(x_1, \dots, x_\ell) \in R^\ell \Leftrightarrow x_1 R x_2 \cdots x_{\ell-1} R x_\ell.$$

Suppose PRM is probabilistically complete. For any query (x, y) , there exists an ℓ such that a sequence of ℓ guesses is a path from x to y with probability $P > 0$. Let $S \subset R^\ell$ be the set of guesses which are length ℓ paths from x to y . The probability of choosing such a sequence of ℓ points is $\mu^\ell(S) = P > 0$ (it can be shown that S is measurable in $\mathcal{Q}_{\text{free}}^\ell$). The set S is decomposed into a union of disjoint rectangles, i.e.,

$$S = \bigcup_{i=1}^{\infty} A_1^i \times \cdots \times A_\ell^i.$$

Choose i such that

$$\mu^n(A_1^i \times \cdots \times A_\ell^i) = \prod_{j=1}^{\ell} \mu(A_j^i)$$

is maximized. Observe that it must be larger than zero. Let x_1, \dots, x_ℓ be any set of points such that $x_j \in A_j^i$. It follows that $x R x_1 R \dots R x_\ell R y$ by construction of S . Let

$$p = \min_j \mu(A_j^i).$$

The probability that PRM fails to find a path between x and y after generating n configurations is therefore bounded by the probability that no such x_1, \dots, x_ℓ is contained in the configuration set. Let I_j be an indicator variable that witnesses the event of the configuration set containing a point from A_j^i .

$$\begin{aligned}
Pr[(x, y) \text{ FAILURE}] &\leq Pr \left[\bigvee_{j=1}^{\ell} I_j = 0 \right] \\
&\leq \sum_{j=1}^{\ell} Pr[I_j = 0] \\
&= \sum_{j=1}^{\ell} (1 - \mu(A_j^i))^n \\
&\leq \ell(1 - p)^n \leq \ell e^{-pn}.
\end{aligned}$$

Finally, note that $\ell \geq 0$ and $p > 0$.

It is interesting to note that the symmetry and reflexivity properties of the local planner were never used in the proof. In particular, the proof will still hold for an asymmetric and irreflexive local planner. This is a natural way to incorporate the notion of time into PRM planning. Also, the sampling distribution is not necessarily uniform. The obtained bound is of the same form as the previous bounds and shows that probabilistic completeness ensures an inverse exponential bound on failure probability in terms of the number of configurations in the roadmap.

 Previous

Next 

TeamUnknown Release