4

# Probabilistic Roadmaps for Robot Path Planning

**Lydia E. Kavraki**[*] **Jean-Claude Latombe**[**]

[*]*Department of Computer Scince, Rice University, Houston, TX 77005*
[**] *Department of Computer Science, Stanford University, Stanford, CA 94305*

**Abstract**

The Probabilistic RoadMap planner (PRM) has been applied with success to multiple planning problems involving robots with 3 to 16 degrees of freedom (dof) operating in known static environments. This paper describes the planner and reports on experimental and theoretical results related to its performance. PRM computation consists of a preprocessing and a query phase. Preprocessing, which is done only once for a given environment, generates a roadmap of randomly, but properly selected, collision-free configurations (nodes). Planning then connects any given initial and final configurations of the robot to two nodes of the roadmap and computes a path through the roadmap between these two nodes. The planner is able to find paths involving robots with 10 dof in a fraction of a second after relatively short times for preprocessing (a few dozen seconds). Theoretical analysis of the PRM algorithm provides bounds on the number of roadmap nodes needed for solving planning problems in spaces with certain geometric properties. A number of theoretical results are presented in this paper under a unified framework.

## 4.1 Introduction

Path planning for robots in known and static workspaces has been studied extensively over the last two decades [Lat91]. Recently there has been renewed interest in developing planners that can be applied to robots with many degrees of freedom (dof), say 5 or more. Indeed,

*Sample Contributed Book*. Editor Jenny Smith

an increasing number of practical problems involve such robots, while very few effective motion planning methods are available to solve them. Besides its use in robotics, planning with many dof is becoming increasingly important in emerging applications, e.g., computer graphic animation, where motion planning can drastically reduce the amount of data input by human animators [KKKL94], and molecular biology, where motion planning can be used to compute motions of molecules (modeled as spatial linkages with many dof) docking against other molecules [Kav97].

In this paper we describe the Probabilistic RoadMap planner (PRM). This planner has been applied with success to multiple planning problems involving robots with 3 to 16 dof moving in static environments. PRM computation is organized in two phases: the *preprocessing phase* and the *query phase*. During the preprocessing phase a probabilistic *roadmap* is constructed by repeatedly generating random free configurations of the robot and connecting these configurations using some simple, but very fast motion planner. We call this planner the local planner. The roadmap thus formed in the free configuration space (C-space [LP83]) of the robot is stored as an undirected graph. The configurations are the nodes of the graph and the paths computed by the local planner are the graph edges. This phase is concluded by some processing, which selectively adds nodes in parts of the free C-space with the goal of improving the roadmap connectivity. Following the preprocessing phase, multiple queries can be answered. A *query* asks for a path between two free configurations of the robot. To answer a query PRM first attempts to find a path from the start and goal configurations to two nodes of the roadmap. Next, a graph search is performed to find a sequence of edges connecting these nodes in the roadmap. Concatenation of the successive path segments transforms the sequence found into a feasible path for the robot. Two snapshots along a path produced by PRM for a rigid 6-dof robot are shown in Figure 4.1.

In this paper we describe the PRM algorithm as this has evolved through our previous work presented in [Kav95, KL94a, KL94b, KŠLO96] and report a number of experimental results. We also discuss the theoretical analysis of the planner developed in [KKL96, KLMR95] and outline a unified framework for this analysis. In particular, we discuss the notion of "goodness" properties [BKL$^+$97] and evaluate the performance of PRM under the assumption of $\epsilon$*-goodness* and *expansiveness* for the solution space and $path$*-goodness* for the planned path.

The organization of the paper is as follows. Section 4.2 gives an overview of some previous research and relates our work to this research. Section 4.3 describes PRM in general terms, i.e., without focusing on any specific type of robot. Both the preprocessing and the query phases are discussed here in detail. Next, in Section 4.4, we present some experiments with rigid and articulated robots. Section 4.5 presents a unified framework for analyzing the performance of PRM and outlines a number of theoretical results. Section 4.6 summarizes the paper.

## 4.2 Relation to Previous Work

Robot path planning has been proven a hard problem [Rei79]. There is strong evidence that its solution requires exponential time in the number of dimensions of the C-space, i.e., the number of degrees of freedom of the robot. The reader is referred to [Lat91, KLMR95] for an overview of the complexity of several complete planners and related hardness results.

Recently there has been renewed interest in developing practical path planners [Lat91, BKL$^+$97]. These planners embed weaker notions of completeness (e.g., probabilistic completeness) and/or can be partially adapted to specific problem domains in order to boost

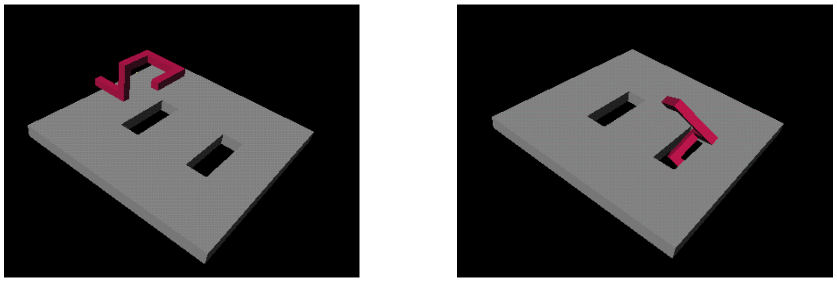

**Figure 4.1** A 6 dof rigid robot moving through a narrow door.

performance in those domains.

Some of the most impressive results have been obtained using potential field methods. Such methods are attractive, since the heuristic function guiding the search for a path, the potential field, can easily be adapted to the specific problem to be solved, in particular the obstacles and the goal configuration. The main disadvantage of these planners is the presence of local minima in the potential fields. These minima may be difficult to escape. Local minima-free potential functions (also called navigation functions) have been defined in [Kod87, RK92, BL91]. But these functions are expensive to compute in high-dimensional C-spaces and have not been used for many-dof robots. One of the first successful potential field planners for robots with many dof is described in [FT89]. This planner employs a learning scheme to avoid falling into the local minima of the potential field and has been applied with some success to robots with up to 6 dof. However the bookkeeping done during the learning phase becomes impractical when the dof grow larger.

Techniques for both computing potential functions and escaping local minima in high-dimensional C-spaces were presented in [BL91, BLL92]. The Randomized Path Planner (RPP) described in [BL91] escapes local minima by executing random walks. An analysis of this planner is initiated in [LL96]. RPP has solved many difficult problems involving robots with 3 to 31 dof. It has also been used in practice with good results to plan motions for performing riveting operations on plane fuselages [GMKL92], and to plan disassembly operations for the maintenance of aircraft engines [CL95]. Additionally, RPP has been embedded in a larger "manipulation planner" to automatically animate scenes involving human figures modeled with 62 dof [KKKL94]. However, several examples have also been identified where RPP behaves poorly [CG93]. In these examples, RPP falls into local minima whose basins of attraction are mostly bounded by obstacles, with only narrow passages to escape. The probability that any random walk finds its way through such a passage is almost zero, while the use of several potential functions that can prevent this from happening is rather time consuming.

Other interesting lines of work include the method in [BF94] which is based on a variational dynamic programming approach and can tackle problems of similar complexity to the problems solved by RPP. In [GG90, GZ94] a sequential framework with backtracking is proposed for serial manipulators, and in [CH92] a motion planner with performance proportional to task difficulty is developed for arbitrary many-dof robots operating in cluttered environments. The planner in [Kon91] finds paths for 6-dof manipulators using heuristic search techniques that limit the part of the C-space that is explored, and the planner in [ATBM92] utilizes genetic algorithms to help search for a path in high-dimensional C-spaces. Parallel processing techniques are investigated in [CG93, LPO91]. Hybrid methods are discussed in [CQM95].

The planning method presented in this paper differs significantly from the methods referenced above, which are for the most part based on potential field or cell decomposition approaches. Instead, PRM applies a roadmap approach [Lat91], that is, it constructs a roadmap of paths in the free C-space. Previous roadmap methods include the visibility graph [LPW79], Voronoi diagram [OY82], and silhouette methods [Can88]. All these three methods compute a roadmap that completely represents the connectivity of the free C-space. The visibility graph and Voronoi diagram methods are limited to low-dimensional C-spaces. The silhouette method applies to C-spaces of any dimension, but its complexity makes it little practical.

Roadmaps have also been built and used incrementally in several other planners. The planner in [CL90] incrementally builds the skeleton of the C-space using a local opportunistic strategy. This work has inspired the approaches in [RC94, CB94] which construct retracts of the free C-space using sensor data and thus do not assume that the (static) environment in which the robot moves is known a priori. The approach in [Che92] builds a sparse network of robot subgoals with the use of a simple and a computationally expensive planner.

The randomized techniques of PRM were initially presented in [KL94a, KL94b]. Parallel independent work was done in [Ove92, ŠO97, OŠ95]. A common paper from the two groups was published in [KŠLO96]. The work in [HST94], again done independently, describes ideas similar to PRM. Applications to non-holonomic robots were investigated in [Šve97, ŠO97]. Applications to many-dof holonomic robots were discussed in [Kav95, KL94a, KL94b, KŠLO96]. The theoretical analysis of PRM was initiated in [KKL96, KLMR95], while [BKL$^+$97] presents PRM in the context of randomized approaches to robot path planning and [HLM97] extends PRM to a new planner that is able to deal with difficult assembly problems.

## 4.3 The PRM Planner

We now describe PRM in general terms for a holonomic robot without focusing on any specific type of robot. During the preprocessing phase a data structure called the roadmap is constructed in a probabilistic way for a given scene. The roadmap is an undirected graph $R = (N, E)$. The nodes in $N$ are a set of configurations of the robot appropriately chosen over the free C-space. The edges in $E$ correspond to (simple) paths; an edge between two nodes corresponds to a feasible path connecting the relevant configurations. These paths, which we refer to as local paths, are computed by an extremely fast, though not very powerful planner, called the local planner. During the query phase, the roadmap is used to solve individual path planning problems in the input scene. Given a start configuration $q_{init}$ and a goal configuration $q_{goal}$, the method first tries to connect $q_{init}$ and $q_{goal}$ to some two nodes $q'_{init}$ and $q'_{goal}$ in $N$. If successful, it then searches $R$ for a sequence of edges in $E$ connecting $q'_{init}$ to $q'_{goal}$. Finally, it transforms this sequence into a feasible path for the robot by recomputing the corresponding local paths and concatenating them.

We assume here that the preprocessing phase is entirely performed before any path planning query. However, the preprocessing and query phases can be interwoven, giving a learning flavor to PRM. For instance, a small roadmap could be first constructed; the roadmap could then be augmented (or reduced) using intermediate data generated while queries are being processed. This interesting possibility will not be explored in the paper, though it is particularly useful to conduct trial-and-error experiments in order to decide how much computation time should be spent in the preprocessing phase.

In our description below, let $\mathcal{C}$ denote the robot's C-space and $\mathcal{C}_{free}$ its free subset (also called the free C-space).

### *4.3.1 The Preprocessing Phase*

The preprocessing phase consists of three successive steps: roadmap construction, roadmap expansion, and roadmap component reduction. The objective of the construction step is to obtain a reasonably connected graph and to make sure that most "difficult" regions in this space contain at least a few nodes. The expansion step is aimed at further improving the connectivity of this graph. It selects nodes of $R$ which, according to some heuristic evaluator, lie in difficult regions of $\mathcal{C}_{free}$ and expands the graph by generating additional nodes in their neighborhoods. Hence, the covering of $\mathcal{C}_{free}$ by the final roadmap depends on the local intricacy of $\mathcal{C}_{free}$. The final step, the component reduction step, is optional and is included here for completeness.

#### Roadmap Construction

Initially the graph $R = (N, E)$ is empty. Then, repeatedly, a random free configuration is generated and added to $N$. For every such new node $q$, we select a number of nodes from the current $N$ and try to connect $q$ to each of them using the local planner. Whenever this planner succeeds to compute a feasible path between $q$ and a selected node $q'$, the edge $(q, q')$ is added to $E$. To make our presentation more precise, let:

- $\Delta$ be a symmetrical function $\mathcal{C}_{free} \times \mathcal{C}_{free} \rightarrow \{0, 1\}$, which returns whether the local planner can compute a path between the two configurations given as arguments;
- $D$ be a function $\mathcal{C} \times \mathcal{C} \rightarrow R^{+} \cup \{0\}$, called the *distance function*, defining a pseudo-metric in $\mathcal{C}$. (We only require that $D$ be symmetrical and non-degenerate.)

The roadmap construction step algorithm can be outlined as follows:

```
1.   $N \leftarrow \emptyset$
2.   $E \leftarrow \emptyset$
3.   **loop**
4.       $q \leftarrow$ a randomly chosen free configuration.
5.       $N_q \leftarrow$ a set of candidate neighbors of $q$ chosen from $N$.
6.       $N \leftarrow N \cup \{q\}$
7.       **forall** $q' \in N_q$, in order of increasing $D(q, q')$ **do**
8.           **if** $\neg$*same_connected_component*$(q, q') \wedge \Delta(q, q')$. **then**
9.               $E \leftarrow E \cup \{(q, q')\}$
10.              Update $R$'s connected components.
```

Several choices for the steps of above algorithm are still unspecified. In particular, we need to define how random configurations are created in step 4, propose a local planner for 8, clarify the notion of a candidate neighbor in 5, and choose the distance function $D$ in 7.

*Creation of random configurations.* The nodes of $R$ should constitute a rather uniform random sampling of $\mathcal{C}_{free}$. Every such configuration is obtained by drawing each of its coordinates from the interval of allowed values of the corresponding dof using the uniform probability distribution over this interval. The obtained configuration is checked for collision. If it is collision-free, it is added to $N$; otherwise, it is discarded.

*The local planner.* Two issues need to be addressed for the local planner. One is whether the planner should be deterministic and the other how fast this planner should be. There are tradeoffs with both issues. If a non-deterministic planner was used, local paths would have to be stored in the roadmap and the roadmap would require more space. Also, a more powerful local planner will be slower than a less powerful one; but because it would be more successful in finding paths, it would require a smaller number of roadmap nodes. Our best experimental results have been obtained when the local planner is deterministic and very fast. These results were also independently confirmed in [Ove92, OŠ95, Šve97].

A quite general such local planner, which is applicable to all holonomic robots, connects any two given configurations by a straight line segment in $\mathcal{C}$ and checks this line segment for collision and joint limits (if any). Verifying that a straight line segment remains within the joint limits is straightforward. Collision checking can be done by recursively decomposing the line segment into two halves and checking for collision the middle configuration until a prespecified resolution has been achieved [BL91].

*The node neighbors.* Another important choice to be made is that of the set $N_q$, the candidate neighbors of $q$. The local planner will be called to connect $q$ with nodes in $N_q$ and the cumulative cost of these invocations dominates preprocessing time. We avoid calls of the local planner that are likely to return failure by submitting only pairs of configurations whose relative distance (according to the distance function $D$) is smaller than some constant threshold `maxdist`. Thus, we define:

$$N_q = \{q' \in N | D(q, q') \leq \texttt{maxdist}\}.$$

Additionally, according to the algorithm outline given above, we try to connect $q$ to all nodes in $N_q$ in order of increasing distance from $q$; but we skip those nodes which are in the same connected component as $q$ at the time the connection is to be tried. By considering elements of $N_q$ in this order we expect to maximize the chances of quickly connecting $q$ to other configurations and, consequently, reduce the number of calls to the local planner (since every successful connection results in merging two connected components into one). In our experiments we found it useful to bound the size of the set $N_q$ by some constant $K$. This additional criterion guarantees that, in the worst case, the running time of each iteration of the main loop of the construction step algorithm is independent of the current size of the roadmap $R$. Thus, the number of calls to the local method is linear in the size of the graph it constructs.

*The distance function.* The function $D$ is used to both construct and sort the set $N_q$ of candidate neighbors of each new node $q$. It should be defined so that, for any pair $(q, q')$ of configurations, $D(q, q')$ reflects the chance that the local planner will *fail* to compute a feasible path between these configurations. One possibility is to define $D(q, q')$ as a measure (area/volume) of the workspace region swept by the robot when it moves along the path computed by the local planner between $q$ and $q'$ in the absence of obstacles. Thus, each local planner would automatically induce its own specific distance function. Since exact computation of swept areas/volumes tends to be rather time-consuming, a rough but inexpensive measure of the swept-region gives better practical results. Very simple distance measures also seem to give good results. For example, when the general local planner described above is used to connect $q$ and $q'$, $D(q, q')$ may be defined as follows:

$$D(q, q') = \max_{x \in \text{robot}} ||x(q) - x(q')||, \tag{4.1}$$

where $x$ denotes a point on the robot, $x(q)$ is the position of $x$ in the workspace when the robot is at configuration $q$, and $||x(q) - x(q')||$ is the Euclidean distance between $x(q)$ and $x(q')$.

**Roadmap Expansion**

If the number of nodes generated during the construction step is large enough, the set $N$ gives a fairly uniform covering of $\mathcal{C}_{free}$. In easy scenes $R$ is then well connected. But in more constrained ones where $\mathcal{C}_{free}$ is actually connected, $R$ often consists of a few large components and several small ones. It therefore does not effectively capture the connectivity of $\mathcal{C}_{free}$.

The purpose of the expansion is to add more nodes in a way that will facilitate the formation of a large component comprising as many of the nodes as possible and will also help cover the more “difficult” (narrow) parts of $\mathcal{C}_{free}$. The identification of these difficult parts of $\mathcal{C}_{free}$ is no simple matter and the heuristic that we propose below clearly goes only a certain distance in this direction.

For each node $q$ found during graph construction, we define a *weight* $w(q)$, which should be large whenever $q$ “is in a difficult region”. Additionally, the weights for all $q$ should add up to $1$. Let us now call the *expansion* of node $q$ the creation at random of another node in the free neighborhood of $q$. The following is then the heuristic *scheme* that we propose. We add a user specified number $M$ of new nodes to our collection. This time instead of choosing them at random, we choose a node among the $N$ ones which the roadmap construction step generated with probability:

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

and we expand that node $q$. The simplest way to expand $q$ is to perform a short random-bounce walk starting from $q$. A random-bounce walk consists of repeatedly picking at random a direction of motion in $\mathcal{C}$ and moving in this direction until an obstacle is hit. When a collision occurs, a new random direction is chosen. And so on. We repeat this until a maximum number of steps have been attained (typically in the order of 45). The final configuration $q'$ reached by the random-bounce walk and the edge $(q, q')$ are inserted into $R$. Moreover, the path computed between $q$ and $q'$ is explicitly stored, since it was generated by a non-deterministic technique. We also record the fact that $q'$ belongs to the same connected component as $q$. Then we try to connect $q'$ to the other connected components of the network in the same way as in the construction step. The expansion step thus never creates new components in $R$. At worst, it fails to reduce the number of components. We repeat the above process $M$ times (good values of $M$ are between $N/2$ and $N$), creating $M$ new nodes in $R$.

If the function $w(q)$ adequately identifies the difficult parts of $\mathcal{C}_{free}$, our heuristic will tend to fill these parts more than others. Thus it is essential to choose $w(q)$ carefully. In our experiments we use the results of roadmap construction to build $w(q)$. We define the *degree* $d_q$ of a node $q$ as the number of connections that $q$ has with other nodes at the end of the roadmap construction. Then:

$$w(q) = \frac{1}{d_q + 1} \Big/ \sum_{t=1}^{N} \frac{1}{d_t + 1}.$$

We regard this number as a measure of the "difficulty" of the C-space region in which the node lies. Nodes with low degree are in "difficult" parts of $\mathcal{C}_{free}$. It is crucial to retain such nodes since they may lie in narrow passages of $\mathcal{C}_{free}$ and may contribute to producing a connected roadmap in $\mathcal{C}_{free}$.

**Component Reduction**

Typically the roadmap expansion step yields one connected component comprising most of the produced nodes, when $N + M$ is large enough. Of course, this is not the case when $\mathcal{C}_{free}$ is not connected. There are also difficult examples where $\mathcal{C}_{free}$ is connected but our local planner failed to achieve some crucial connections.

There are difficult examples where a few large components remain at the end of graph expansion. Then we have two choices. The first is to continue adding nodes until a maximum amount of time has elapsed. This is our preferred course of action and in most problems PRM will quickly find a large connected component of $R$ that captures the connectivity of the underlying space. The second option, presented here for completeness, is to use a powerful planner to attempt connections among any remaining components.

We can proceed as follows. For any two components, starting with the largest, we select a pair of nodes, $p$ in the first and $q$ in the second, which are close to each other (according to the C-space metric $D$). In our implementation RPP [BL91] is called to connect $p$ and $q$. If it produces a connection, the two connected components are merged into one, and a new pair of components is considered. Instead, if it fails to produce a connection within some short time bound, a different pair of $p$ and $q$ is chosen and a new connection is attempted. In this way we avoid getting stuck at a case that may be hard for RPP, or even impossible (if the two components are not path-connected in $\mathcal{C}_{free}$). The process is repeated a few times for each pair of components. Eventually, if RPP fails to produce a connection, we consider that the components cannot be connected and we retain them as they are. Connection paths computed during that step are recorded, since their recomputation would be relatively expensive.

### *4.3.2 The Query Phase*

During the query phase, paths are to be found between arbitrary input start and goal configurations $q_{init}$ and $q_{goal}$, using the roadmap constructed in the preprocessing phase. Assume for the moment that $\mathcal{C}_{free}$ is connected and that the roadmap consists of a single connected component $R$. We try to connect $q_{init}$ and $q_{goal}$ to some two nodes of $R$, respectively $q'_{init}$ and $q'_{goal}$, with feasible paths $P_{init}$ and $P_{goal}$. If this fails, the query fails. Otherwise, we compute a path $P$ in $R$ connecting $q'_{init}$ to $q'_{goal}$. A feasible path from $q_{init}$ to $q_{goal}$ is eventually constructed by concatenating $P_{init}$, the recomputed path corresponding to $P$, and $P_{goal}$ reversed. This path may be improved by running a smoothing algorithm on it. Possible smoothing techniques include the one in [LTJ90], which selects random segments of the global path and tries to shortcut them by using the local planner, and the method in [BG94], which iteratively performs local geometric operations (i.e., cutting off triangle corners).

The main question is how to compute the paths $P_{init}$ and $P_{goal}$. The queries should preferably terminate quasi-instantaneously, so no expensive algorithm is desired here. Our strategy for connecting $q_{init}$ to $R$ is to consider the nodes in $R$ in order of increasing distance from $q_{init}$ (according to $D$) and try to connect $q_{init}$ to each of them with the local planner, until one connection succeeds or until an allocated amount of time has elapsed. If all connection attempts fail, we can perform one or more random-bounce walks, as described in Section 4.3.1. But, instead of adding the node at the end of each such random-bounce walk to the roadmap, we now try to connect it to $R$ with the local planner. As soon as $q_{init}$ is successfully connected to $R$, we apply the same procedure to connect $q_{goal}$ to $R$. The reconstruction of a robot path from the sequence of nodes in $P$ reduces to the concatenation of the paths that take the robot between adjacent nodes in $P$. Some of these paths have been produced by random-bounce walks during the preprocessing phase and are stored in the relevant edges of $R$. Paths corresponding to connections that have been found by the local planner are recomputed.

In general, however, the roadmap may consist of several connected components $R_i$, $i = 1, 2, \ldots, k$. This is usually the case when $\mathcal{C}_{free}$ is itself not connected. It may also happen when $\mathcal{C}_{free}$ is connected, for instance if the roadmap is not dense enough. If the roadmap contains several components, we try to connect both $q_{init}$ and $q_{goal}$ to two nodes in the *same* component, starting with the component closest to $q_{init}$ and $q_{goal}$. If the connection of $q_{init}$ and $q_{goal}$ to some component $R_i$ 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. Since in most examples the roadmap consists of rather few components, failure is rapidly detected.

If path planning queries fail frequently, this is an indication that the roadmap may not adequately capture the connectivity of $\mathcal{C}_{free}$. Hence, more time should be spent in the preprocessing phase, i.e., $N+M$ should be increased. However, it is not necessary to construct a new roadmap from the beginning. Since the preprocessing phase is incremental, we can simply extend the current roadmap by resuming the construction step algorithm and/or the expansion step algorithm, starting with the current roadmap graph, thus interweaving the preprocessing and the query phases.

## 4.4 Experimental Results

We demonstrate the application of PRM to one example in detail and briefly report on its performance with a couple of other examples. Other experiments can be found in [Kav95, KL94a, KL94b, KŠLO96]. The planner is implemented in C and for the experiments reported here we used a DEC Alpha workstation. This machine is rated at 126.0 SPECfp92 and 74.3 SPECint92.

### A 2D articulated robot

We use as a first example a planar articulated robot with an arbitrary number of revolute joints. Figure 4.2 illustrates such a robot in which the links are line segments. The shape of the links may actually be any polygon. Points $J_1$ through $J_k$ (in the figure, k=6) designate revolute joints. Point $J_1$ denotes the base of the robot; it may, or may not, be fixed relative to the workspace. The point $J_k$ ($J_6$ in the figure) is called the endpoint of the robot. Each of the rest of the revolute joints $J_i$ has defined internal joint limits, denoted by $low_i$ and $up_i$, with $low_i < up_i$. If the robot's base is free, the translation of $J_1$ is bounded along the $x$ and $y$ axes of the Cartesian coordinate system embedded in the workspace

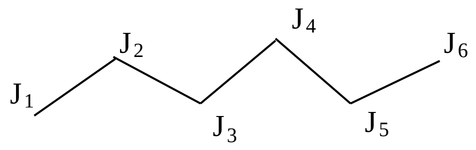

**Figure 4.2** A planar articulated robot.

by $low_x$ and $up_x$, and $low_y$ and $up_y$, respectively. We represent the C-space of such a $k$-link planar articulated robot by $[low_1, up_1] \times \ldots \times [low_k, up_k]$, if its base is fixed, and by $[low_x, up_x] \times [low_y, up_y] \times [0, 2\pi] \times [low_2, up_2] \times \ldots \times [low_k, up_k]$, if its base is free. A self-collision configuration is any configuration where two non-adjacent links of the robot intersect each other. We do not allow such configurations. Thus, $\mathcal{C}_{free}$ is constrained by the obstacles and by the set of self-collision configurations. We outline below our choices for the implementation of PRM when these are different from the ones described in Section 4.3.

*The local planner.* The planner of Section 4.3 attempts to connect two configurations by a straight line in $\mathcal{C}_{free}$ and for articulated robots gives reasonably good results. However, a planner that proved better for articulated robots is the following. Let $p$ and $q$ be the two configurations we attempt to connect and $J_1, \ldots, J_k$ be the points on the robot as shown in Figure 4.2. We simultaneously translate every second $J_i$, that is $J_{2*i+1}$, $i = 0, \ldots, k/2$, along the straight line in the workspace that connects its workspace position at configuration $p$ to its workspace position at configuration $q$. Then we adjust the positions of $J_{2*i}$, $i = 1, \ldots, k/2$, by computing the inverse kinematics of the robot. In this way the $J_{2*i}$'s follow the motion initiated by the $J_{2*i+1}$'s. If $k$ is even then the position of $J_k$ is not determined by the above. It can be determined by moving the last dof of the robot (the one that specifies the position of $J_k$) on a straight line in $\mathcal{C}$ so that it reaches its desired value in configuration $q$. If during the motion, the robot collides with an obstacle or with itself, or if a joint reaches a limit, or an adjustment is impossible, the planner fails. The planner generalizes directly to 3D articulated robots and any kind of robot for which we can move a few of its dof at a certain direction and adjust the others.

*The distance function.* Let $J_i(q)$, $i = 1, \ldots, k$ denote the position of $J_i$ (see Figure 4.2), when the robot is at configuration $q$. We define the distance $D(q, q')$ between any two configurations $q$ and $q'$ as:

$$D(q, q') = \left( \sum_{i=1}^{k} ||J_i(q) - J_i(q')||^2 \right)^{1/2},$$

where $||J_i(q) - J_i(q')||$ is the Euclidean distance between $J_i(q)$ and $J_i(q')$. This distance is quick to compute and has an intuitive meaning for articulated robots.

*Collision checking.* Collision checking for planar robots can be implemented using a discretized C-space bitmap for each link of the robot. We assume that each link of the robot is free to translate and rotate and we precompute for it a 3D C-space bitmap that explicitly represents the free subset of the link's C-space (the "0"s) versus the part that gives rise to collision with an obstacle (the "1"s). When testing for collision, PRM tests each link against its C-space bitmap, which is very fast. This technique is practical only for 2D workspaces, since 3D workspaces would require the generation of 6D bitmaps. For our examples we discretize each dof to 128 values and we compute the C-space bitmaps with the use of the Fast Fourier

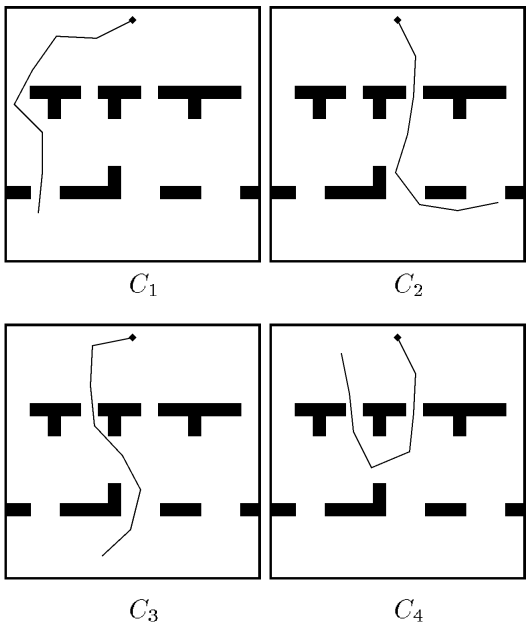

**Figure 4.3** A 2D articulated robot with a fixed base and 7 revolute joints.

| $\mathtt{T}_{total}$ (sec) | $\mathtt{T}_{constr}$ (sec) | $\mathtt{T}_{exp}$ (sec) | Collision checks | Avg. nodes | Success Rate (%) $C_1$ | $C_2$ | $C_3$ | $C_4$ |
|---|---|---|---|---|---|---|---|---|
| 20.1 | 13.1 | 7.0 | 621943 | 1062 | 100.0 | 36.7 | 56.7 | 36.7 |
| 30.1 | 19.5 | 10.6 | 889384 | 1643 | 100.0 | 66.7 | 70.0 | 66.7 |
| 40.3 | 26.3 | 14.0 | 1145091 | 2233 | 100.0 | 90.0 | 86.7 | 90.0 |
| 50.3 | 32.7 | 17.6 | 1392454 | 2783 | 100.0 | 96.7 | 96.7 | 96.7 |
| 60.2 | 39.1 | 21.1 | 1631612 | 3284 | 100.0 | 100.0 | 100.0 | 100.0 |
| 70.3 | 45.8 | 24.5 | 1876006 | 3805 | 100.0 | 96.7 | 100.0 | 96.7 |
| 80.4 | 52.2 | 28.2 | 2104209 | 4272 | 100.0 | 100.0 | 100.0 | 100.0 |

**Figure 4.4** PRM applied to the 2D 7-revolute-joint robot.

Transform [Kav93]. For self-collision, each link of the robot is tested against the others.

*Experiments.* Figure 4.3 shows four configurations forming the test set of an articulated robot in a scene with several narrow gates. The robot has a fixed base, denoted by a square, and 7 revolute dof.

The table in Figure 4.4 reports the success rates of connecting the configurations in the test set to roadmaps obtained with different preprocessing times. The preprocessing time, $\mathtt{T}_{total}$, is shown in column 1. It is broken into $\mathtt{T}_{constr}$ and $\mathtt{T}_{exp}$, the times spent in roadmap construction and expansion, in columns 2 and 3 respectively (note that $\mathtt{T}_{exp}$ is approximately $1/3$ of $\mathtt{T}_{total}$). Also, $\mathtt{maxdist} = 0.4$ (the workspace is $[0,1] \times [0,1]$), $K = 30$, and the number of steps of the random bounce are 45.

For every row in Figure 4.4 we independently generated 30 roadmaps, each with the indicated preprocessing time. The roadmaps generated for different rows were also computed

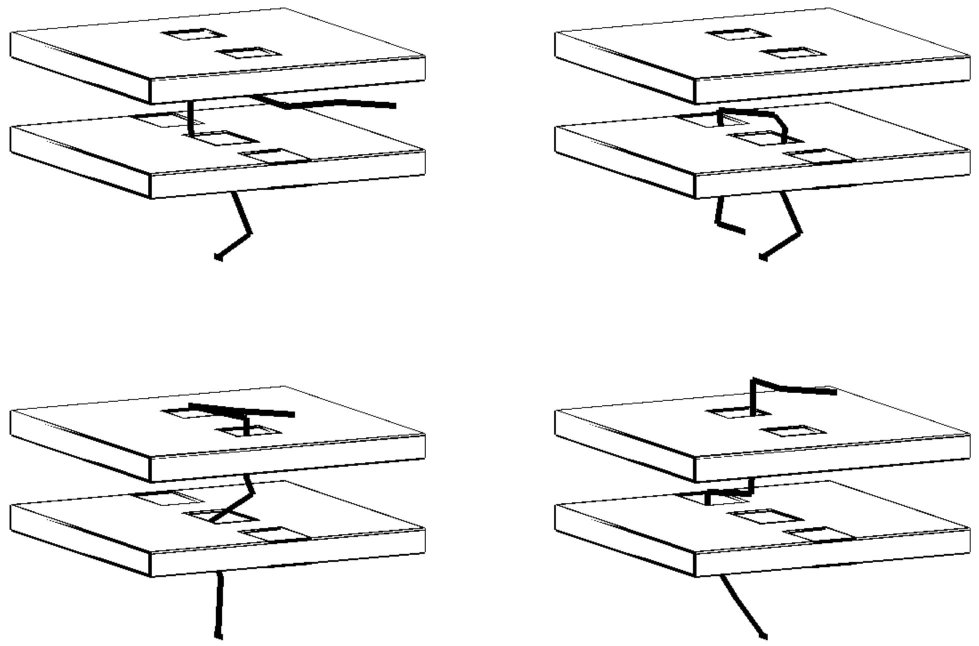

**Figure 4.5** Fixed-base robot with 8 spherical joints (16 dof).

independently, that is, no roadmap in some row was reused to construct a larger one in the following row. Column 4 in Figure 4.4 gives the average number of collision checks performed during the preprocessing phase for different preprocessing times. Column 5 in Figure 4.4 reports the average number of nodes, over the 30 runs, in the largest roadmap component at the end of the preprocessing phase. The largest connected component of each roadmap is used for query processing. Columns 6 though 9 are labeled with the four configurations $C_1, \ldots, C_4$ of Figure 4.3. The columns report the success rate when trying to connect, in less that 2.5 seconds, the corresponding configuration to each of the 30 produced roadmaps. One trial was made per roadmap. Note that after a preprocessing of 50 sec, the success rate of connecting each of $C_1, C_2, C_3, C_4$ to the roadmap is above $96.6\%$. Path planning will succeed between any two configurations that can be connected to the roadmaps produced as described in Section 4.3. Typically it took a small fraction of second to reconstruct the path in the machine used.

### A 3D articulated robot

We show in Figure 4.5 four configurations of a 16-dof articulated robot. The robot has a fixed base and 8 spherical joints. The creation of the random configurations, and the distance function used are the same as in the 2D case. However, the local planner is a straight-line planner and collision checking is done by checking each of the links of the robot against all

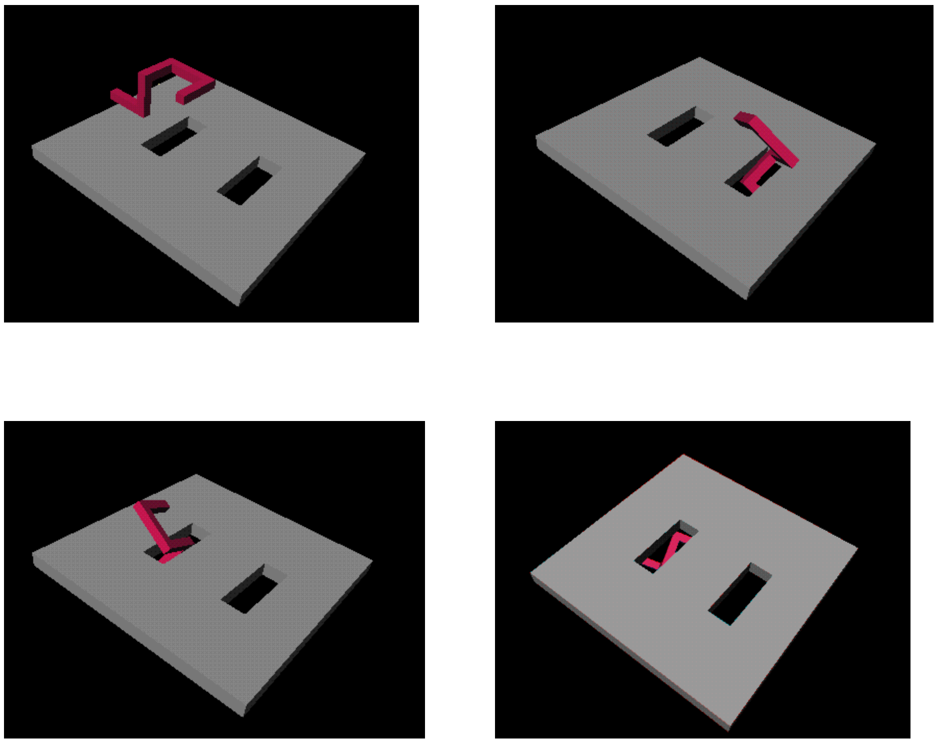

**Figure 4.6** Rigid robot with 6 dof.

the obstacles. For this example we conducted similar experiments as and in the 2D case and we found that approximately 769 seconds are needed to produce a roadmap that adequately captures $\mathcal{C}_{free}$. That roadmap contained approximately $4,700$ nodes and the success rate for the queries was $90\%$ or higher, again when allowing 2.5 seconds to process a query.

### A 3D rigid robot

Another example is shown in Figure 4.6. This is a rigid robot which needs to go through a small gate in its workspace. The robot in this case consists of five polyhedral links of the same size which are rigidly attached to each other. The workspace is such that the robot needs to rotate while going through the gate. The width of the gate determines the difficulty of the problem. In our implementation the gate is 2.5 times wider than the smallest dimension of each robot link. Our implementation choices remain the same as in the 2D case except that the local planner is the straight-line planner and that collision checking was performed using RAPID [LMCG95].

Similar experiments over a large number of runs have shown that approximately 1.5 hours are required to create a roadmap to which all of the configurations in Figure 4.6 can be connected with probability higher than $90\%$. The roadmap contained $20,000$ nodes on the average. We also note that preprocessing time drops to a fews tens of minutes once the workspace gate is 5 times wider than the smallest dimension of each robot link.

## 4.5 Theoretical Analysis

The intuitive reason for the experimental success of PRM is that there usually exist many collision-free paths joining two configurations. Hence, to bound the running time of PRM we assume that $\mathcal{C}_{free}$ satisfies some geometric property capturing the above intuition. We propose three such properties. Our thesis is that the success of any sampling strategy will stem from a similar property.

In our analyses, we consider that the key number affecting PRM's running time is the number of nodes in the constructed roadmap. Since to generate a single node it might be necessary to randomly pick several (possibly, many) configurations in $\mathcal{C}$, we implicitly assume that the volume of $\mathcal{C}_{free}$ relative to the volume of $\mathcal{C}$ is not too small. If this assumption is not satisfied, any variant of the roadmap algorithm presented in Section 4.3 will behave poorly. In this case, we should probably make the additional assumption that a few free configurations are given (after all, every query will specify two such configurations); then, a possible sampling strategy could be to build a roadmap by generating nodes in small regions centered at the given configurations, first, and at the newly generated nodes, next.

Note that bounding the number of nodes is not sufficient to bound the running time of PRM, because it does not account for the running time of the local planner. Suppose that two nodes can be connected by the local planner but the path connecting them is arbitrarily close to the free space boundary. Then the local planner may have to break it into arbitrarily many small segments [BKL$^+$97]. This problem could be eliminated by computing the volume swept out by the robot when it moves between two configurations along a straight path in $\mathcal{C}$, but this computation can be very expensive.

### *4.5.1 Basic-PRM*

For our analysis we rid the planner of choices that were made to improve its performance and do not consider the roadmap expansion step of the preprocessing. We call this new planner basic-PRM.

Let $s$ be the nodes in the constructed roadmap. We say that configuration $q$ can "see" configuration $q'$ if it can be connected to it by a local planner. The preprocessing step is as follows:

```
Preprocessing:
1. $i \leftarrow 0$.
2. While $i < s$ do:
   (a) Pick a configuration $q$ in $\mathcal{C}$ at random.
   (b) If $q$ is in $\mathcal{C}_{free}$ then
       i. Store $q$ as a node of the roadmap.
       ii. $i \leftarrow i + 1$.
3. For every pair of nodes call the local planner.
4. Pick one representative node from each component of the current roadmap.
   Let $V$ be the set of these representative nodes. Invoke Permeation($V$) to
   improve the connectivity of the nodes (see below).
```

Note that steps 1-3 are as in the roadmap construction step of PRM in Section 4.3, while step 4 corresponds to the roadmap component reduction step of Section 4.3. The result of the preprocessing should be a roadmap $R$ such that no two components of $R$ are in the same component of $\mathcal{C}_{free}$. Step 3 may fail to find all possible links between the nodes due to the incompleteness of the local planner. `Permeation` invoked in step 4 fixes this problem by using the complex planner to discover additional connections between nodes. We assume that this complex planner is error-free in that it discovers a path between two given configurations whenever one exists, and reports failure when there is none. But of necessity such a complete planner is expensive to run, and so we seek to use it sparingly.

The query processing is handled by the following algorithm:

```
 Query-Processing($q_{init}, q_{goal}$):
1. For $i = \{init, goal\}$ do:
   (a) If there exists a node m that sees $q_i$ then $m_i \leftarrow m$.
   (b) Else
       i. Repeat $g$ times:
          Pick a configuration q uniformly at random in a neighborhood of $q_i$
          until q sees both $q_i$ and a node m.
      ii. If all $g$ trials failed then return FAILURE and halt, else $m_i \leftarrow m$.
2. If $m_{init}$ and $m_{goal}$ are in the same component of the roadmap then return YES;
   else return NO.
```

Step 1(b)i differs slightly from the corresponding step of the query-processing algorithm in Section 4.3. Picking configurations in this set requires guessing configurations in C-space and retaining only those that are seen by $q_i$. This means that each of the $g$ iterations performed at Step 1(b)i may require several trials to obtain a configuration visible from $q_i$. The analysis proposed below ignores these trials and focuses only on the number $g$. Another difference between the above algorithm and the one of Section 4.3 is that it returns FAILURE (instead of NO) at Step 1(b)ii. Due to the use of the `Permeation` in the preprocessing, both the answers YES and NO are now always correct. With some probability though, the query-processing algorithm may fail to give an answer.

### 4.5.2 *Visibility Assumption: $\epsilon$-goodness*

For any configuration $q \in \mathcal{C}_{free}$, let $\mathcal{V}(q)$ consist of all those configurations $q' \in \mathcal{C}_{free}$ that $q$ sees. We denote the volume of a subset $\mathcal{X}$ of $\mathcal{C}$ by $\mu(\mathcal{X})$.

**Definition 4.5.1** *Let $\epsilon$ be a positive real. A configuration $q \in \mathcal{C}_{free}$ is $\epsilon$-good if $\mu(\mathcal{V}(q)) \geq \epsilon\mu(\mathcal{C}_{free})$. Furthermore, $\mathcal{C}_{free}$ is $\epsilon$-good if all the configurations it contains are $\epsilon$-good.*

The visibility volume assumption made here is that $\mathcal{C}_{free}$ is $\epsilon$-good, that is, each configuration in it sees a significant portion of $\mathcal{C}_{free}$. The underlying intuition is that it is then relatively easy to pick a set of nodes that, collectively, can see most of $\mathcal{C}_{free}$. But the assumption fails to prevent $\mathcal{C}_{free}$ from containing narrow passages through which it might be difficult to connect nodes. For example, consider the case where $\mathcal{C}$ is two-dimensional and $\mathcal{C}_{free}$ consists of two disks of equal size that overlap by a very small amount. Then $\mathcal{C}_{free}$ is $\epsilon$-good for $\epsilon \approx 0.5$. But the probability that any node in one disk sees a node in the other disk is very small. For this reason, we need the third step of the preprocessing of PRM which employs a powerful complex planner.

Let us now present some performance guarantee for the preprocessing phase. Call a set of nodes *almost complete* if the volume of the subset of $\mathcal{C}_{free}$ not visible from any of these nodes is at most $(\epsilon/2)\mu(\mathcal{C}_{free})$. Intuitively, if we were to place a point source of light at each node, we would like a fraction at least $1-\epsilon/2$ of $\mathcal{C}_{free}$ to be illuminated.

**Theorem 4.5.1** *Assume that $\mathcal{C}_{free}$ is $\epsilon$-good. Let $\phi$ be a constant in $(0,1]$ and $K$ be a positive real large enough that for any $x \in (0,1]$, $(1-x)^{(K/x)\log(2/x\phi)} \leq x\phi/2$. If $s$ is chosen such that:*

$$s \geq \frac{K}{\epsilon}(\log\frac{1}{\epsilon} + \log\frac{2}{\phi}),$$

*then* `preprocessing` *generates an almost complete set of nodes for $\mathcal{C}_{free}$ with probability at least $1-\phi$.*

For the proof of this theorem see [KLMR95]. Note that the probability that a roadmap does not provide almost complete coverage of $\mathcal{C}_{free}$ decreases exponentially with the number of nodes. Also, as $\epsilon$ increases, the requirement for an almost complete set grows weaker, i.e., the portion of the free space that has to be visible by at least one node gets smaller. Intuitively, this comes from the fact that a greater $\epsilon$ will make it easier to connect query configurations to the roadmap. Naturally, the number of nodes needed becomes smaller. Theorem 4.5.1 only says that most of $\mathcal{C}_{free}$ is likely to be visible from some node in the roadmap; using this property alone, we can show that queries can be answered quickly [KLMR95]:

**Theorem 4.5.2** *Let the number of iterations $g$ at Step 1(b)i of* `query-processing` *be set to $\log(2/\psi)$, where $\psi$ is a constant in $(0,1]$. If an almost complete set of nodes has been chosen, then the probability that* `query-processing` *outputs* FAILURE *is at most $\psi$.*

In order to ensure a good probability that the query processing outcomes YES or NO, we need to use the complex planner in `Permeation` to determine which nodes of $V$ are reachable from each other. By calling the complex planner we partition $V$ into subsets such that all the nodes in the same subset belong to the same component of $\mathcal{C}_{free}$ and no two nodes in two different subsets are in the same component of $\mathcal{C}_{free}$. If $v$ is the size of $V$, this can be done with $O(v^2)$ invocations of the complex planner by trying it on every pair of nodes in $V$. In [KLMR95] we present a randomized algorithm for solving permeation. Suppose there are $k$ components in $V$. The analysis of the algorithm in [KLMR95] shows that the worst case is when all components are equal to $v/k$ and then the expected cost of `permeation` is $O(vk)$. When there is one giant component and $k-1$ components of size $O(1)$ the expected cost is $\Theta(v+k^2)$, which is shown to be the non-deterministic lower bound.

### *4.5.3 Visibility Assumption: Roadmap Connectedness*

The $\epsilon$-goodness assumption does not prevent the existence of narrow passages in $\mathcal{C}_{free}$. In the presence of narrow passages, `preprocessing` may require a considerable amount of time to build a connected roadmap due to the calls to the complex planner. Intuitively what happens is that a very small subset of $\mathcal{C}_{free}$ at the one end of the passage sees a large fraction of $\mathcal{C}_{free}$ at the other end of the passage. Therefore, the probability that `preprocessing` will pick nodes outside the passage that see each other is small. Similarly, the probability of picking a node inside the passage decreases as the passage gets narrower.

To remove the need for the "complex planner" we now introduce the notion of *expansiveness*. We first define precisely the kind of roadmap we would like `preprocessing` to construct.

**Definition 4.5.2** *Let $\mathcal{C}_{free}$ be an $\epsilon$-good space. A roadmap $R$ is an* adequate representation *of $\mathcal{C}_{free}$ if its nodes provide an almost complete coverage of $\mathcal{C}_{free}$ and no two components of $R$ lie in the same component of $\mathcal{C}_{free}$.*

Let $R$ be an adequate representation of $\mathcal{C}_{free}$. Since $\mathcal{C}_{free}$ is $\epsilon$-good, no component of $\mathcal{C}_{free}$ has volume less than $\epsilon\mu(\mathcal{C}_{free})$. Therefore, at least one node of $R$ lies in every component of $\mathcal{C}_{free}$ . Since no two components of $R$ lie in the same component of $\mathcal{C}_{free}$ , there is a one-to-one correspondence between the components of $R$ and those of $\mathcal{C}_{free}$.

Let us refer to the subset of points in $S \subset \mathcal{C}_{free}$ that can see a large portion of $\mathcal{C}_{free} \setminus S$ as the lookout of $S$. The previous example suggests that we characterize narrow passages by the minimum volume of the lookout of the visibility set $\mathcal{V}(q)$ over all $q \in \mathcal{C}_{free}$. If a set $\mathcal{V}(q)$ has a small lookout, `preprocessing` will have difficulty computing a roadmap capturing the connectivity of $\mathcal{C}_{free}$ without calling the complex planner.

**Definition 4.5.3** *Let $\beta$ be a constant in $(0, 1]$ and $S$ be a subset of a component $E$ of the free space $\mathcal{C}_{free}$. The $\beta$-*lookout *of $S$ is the set:*

$$\beta\text{-LOOKOUT}(S) = \{q \in S \,|\, \mu(\mathcal{V}(q)\setminus S) \geq \beta \times \mu(E\setminus S)\}.$$

**Definition 4.5.4** *Let $\epsilon$, $\alpha$, and $\beta$ be constants in $(0, 1]$. The free space $\mathcal{C}_{free}$ is $(\epsilon, \alpha, \beta)$-*expansive *if it is $\epsilon$-good and for every $q \in \mathcal{C}_{free}$ we have:*

$$\mu(\beta\text{-LOOKOUT}(\mathcal{V}(q))) \geq \alpha \times \mu(\mathcal{V}(q)).$$

Given the above definition for expansiveness we compute an upper bound for the number of nodes that are needed to build an adequate roadmap without calling the complex planner.

**Theorem 4.5.3** *Assume that $\mathcal{C}_{free}$ is $(\epsilon, \alpha, \beta)$-expansive. Let $\xi$ be a constant in $(0, 1]$. If $s$ is chosen such that:*

$$s \geq \frac{16}{\epsilon\alpha} \log \frac{8}{\epsilon\alpha\xi} + \frac{6}{\beta} + 4,$$

*then* `preprocessing` *generates a roadmap that is an adequate representation of $\mathcal{C}_{free}$ with probability at least $1 - \xi$.*

The proof of this theorem is given in [HLM97]. Its significance is twofold. The probability that a roadmap does not adequately represent $\mathcal{C}_{free}$ decreases exponentially with the number of nodes, and the number of nodes needed increases moderately when $\epsilon$, $\alpha$, and $\beta$ decrease.

## *4.5.4 Path Clearance Assumption*

Another assumption that removes the need for a complex planner is the *path clearance* assumption: we assume that between the two configurations given by a query, there exists a collision-free path $\tau$ of length $L$ that achieves some clearance $\sigma$ between the robot and the obstacles.

More formally, let us parametrize $\tau$ by the arc length $\ell$ from the initial configuration and let $L$ designate the path's total length, i.e., $\tau : \ell \in [0, L] \rightarrow \tau(\ell) \in \mathcal{C}_{free}$. We define $\sigma(\ell)$ to be the Euclidean distance between $\tau(\ell)$ and the free space boundary, and $\sigma_{inf}$ to be $\inf_{\ell \in [0,L]} \sigma(\ell)$. We consider the version of basic-PRM as outlined in Section 4.5.1 with the exception of the permeation step. The query-processing algorithm is also simpler than what we considered in Section 4.5.1, in that it only checks if the initial and goal configurations see nodes in the roadmap. If any one of these connections fails, the query-processing algorithm returns NO. Hence, this query-processing algorithm is deterministic.

Under the path clearance assumption, any NO outcome is incorrect. Let $\zeta$ be the probability that we are willing to tolerate for this event. The following two theorems relate the size of the roadmap to this probability, as well as to the two parameters of the path-clearance assumption, that is, the length $L$ of the hypothesized path and its clearance. They give a performance guarantee for the whole planner.

**Theorem 4.5.4** *Let $\zeta \in (0, 1]$ be a positive real constant. Let $a$ be the constant $2^{-n}\mu(\mathcal{B}_1)/\mu(\mathcal{C}_{free})$ where $\mathcal{B}_1$ denotes the unit ball in $\mathbb{R}^n$. If $s$ is chosen such that:*

$$\frac{2L}{\sigma_{inf}} \exp(-a\sigma_{inf}^n s) \leq \zeta, \tag{4.2}$$

*then the planner outputs* YES *with probability at least $1 - \zeta$.*

Note that, for any given $L$ and $\sigma_{inf}$, the quantity on the left side of the above inequality tends toward zero when $s \rightarrow \infty$. It is even more important to remark that it depends exponentially on $s$.

The following theorem is similar to the previous one, but makes use of the clearance distribution $\sigma(\ell)$ rather than just its infimum:

**Theorem 4.5.5** *Let $\zeta \in (0, 1]$ be a positive real constant. Let $a$ be the same constant as in Theorem 4.5.4. If $s$ is chosen such that:*

$$6 \int_0^L \frac{\exp(-a2^{-n}\sigma^n(\ell)s)}{\sigma(\ell)} d\ell \leq \zeta, \tag{4.3}$$

*then the planner outputs* YES *with probability at least $1 - \zeta$.*

We refer the reader to [KKL96] for the proof of both theorems above. Relations 4.2 and 4.3 imply that the number of nodes that the planner must generate to output YES with probability at least $1 - \zeta$ is polynomial in $1/\sigma_{inf}$ and logarithmic in $L$.

## 4.6 Conclusion

We have described PRM, a two-phase method for solving robot motion planning problems in static workspaces. In the preprocessing phase, PRM constructs a probabilistic roadmap as a collection of configurations randomly selected across the free C-space. In the query phase, it uses this roadmap to quickly process path planning queries, each specified by a pair of configurations. The preprocessing phase includes a heuristic evaluator to identify difficult regions in the free C-space and increase the density of the roadmap in those regions. This feature enables us to construct roadmaps that capture well the connectivity of the free C-space. The method is general and can be applied to virtually any type of holonomic robot as described in this paper. Applications to non-holonomic robots can be found in [Šve97, ŠO97]. Finally, the method can be regarded as a learning approach by interleaving the preprocessing and query processing phases.

The analyses of PRM relate geometric properties of the solution space to the performance of the planner. We provided performance guarantees for PRM under the assumption of $\epsilon$-*goodness* and *expansiveness* for the solution space, and $path$-*goodness* for the planned path. Much work still has to be done to understand the role of the key parameters of PRM and find out how to adjust them automatically. The expansion step of the preprocessing phase also deserves closer attention. Our hope is that the analysis of PRM will further contribute to explaining the excellent experimental results of PRM and suggest sampling techniques that will improve the performance of the planner.

A challenging goal would also be to extend the method to dynamic scenes. One first question is: how should a roadmap computed for a given workspace be updated if a few obstacles are removed or added? Another question worth exploring is how the method can incorporate dynamic or other constraints for the motion of the robot.

**Acknowledgments** This work has been supported by grants DAAA21-89-C0002, N00014-92-J-1809, N00014-94-1-0721 and DAAH04-96-1-1007. L. Kavraki is currently partially supported by NSF CAREER Award IRI-970228. Parts of this paper have been published in [Kav95, KL94a, KL94b, KKL96, KLMR95, KŠLO96, BKL$^+$97]. The authors would like to thank their collaborators in the course of this work, especially Michael Kolountzakis, Rajeev Motwani, Mark Overmars, Praghavan Raghavan, and Petr Švestka. They would also like to thank Bruce Donald, Leo Guibas, Danny Halperin, David Hsu, Steve Lavalle, and Jean-Paul Laumond for many useful discussion and comments.

## REFERENCES

[ATBM92] Ahuactzin J. M., Talbi E.-G., Bessière P., and Mazer E. (1992) Using genetic algorithms for robot motion planning. In *10th Europ. Conf. Artific. Intelligence*, pages 671–675. London, England.

[BF94] Barraquand J. and Ferbach P. (1994) Path planning through variational dynamic programming. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1839–1846. San Diego, CA.

[BG94] Berchtold S. and Glavina B. (1994) A scalable optimizer for automatically generated manipulator motions. In *Proc. IEEE/RSJ/GI Int. Conf. Intelligen Robots and Systems*, pages 1796–1802. München, Germany.

[BKL$^+$97] Barraquand J., Kavraki L., Latombe J., Li T.-Y., Motwani R., and Raghavan P. (1997) A random sampling scheme for path planning. *Int. J. of Robotics Research* To appear.

[BL91] Barraquand J. and Latombe J. (1991) Robot motion planning: A distributed representation approach. *Int. J. of Robotics Research* 10: 628–649.

[BLL92] Barraquand J., Langlois B., and Latombe J.-C. (1992) Numerical potential field techniques for robot path planning. *IEEE Tr. Syst., Man, and Cybern.* 22(2): 224–241.

[Can88] Canny J. (1988) *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA.

[CB94] Choset H. and Burdick J. (1994) Sensor based planning and nonsmooth analysis. In *Proc. of IEEE Int. Conf. Robotics and Automation*, pages 3034–3041. San Diego, CA.

[CG93] Chalou D. and Gini M. (1993) Parallel robot motion planning. In *Proc. of IEEE Int. Conf. Robotics and Automation*, pages 24–51. Atlanta GA.

[CH92] Chen P. and Hwang Y. (1992) Sandros: A motion planner with performance proportional to task difficulty. In *Proc. of IEEE Int. Conf. Robotics and Automation*, pages 2346–2353. Nice, France.

[Che92] Chen P. (1992) Improving path planning with learning. In *Proc. Machine Learning Conference*, pages 55–61.

[CL90] Canny J. and Lin M. (1990) An opportunistic global path planner. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1554–1559. Cincinnati, OH.

[CL95] Chang H. and Li T. (1995) Assembly maintainability study with motion planning. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, pages 1012–1019.

[CQM95] Cameron S., Qin C., and McLean A. (1995) Towards efficient motion planning for manipulators with complex geometry. In *Proc. of the IEEE Int. Symposium on Assembly and Task Planning*, pages 207–212.

[FT89] Faverjon B. and Tournassoud P. (1989) A practical approach to motion planning for manipulators with many degrees of freedom. In Miura H. and Arimoto S. (eds) *Robotics Research 5*, pages 65–73. MIT Press.

[GG90] Gupta K. and Guo Z. (1990) Motion planning with many degrees of freedom: sequential search with backtracking. *IEEE Transactions on Robotics and Automation* 6(5): 522–532.

[GHLW95] Goldberg K., Halperin D., Latombe J., and Wilson R. (eds) (1995) *Algorithmic Foundations of Robotics*. A K Peters, Ltd.

[GMKL92] Graux L., Millies P., Kociemba P., and Langlois B. (1992) Integration of a path generation algorithm into off-line programming of airbus panels. TP 922404, SAE.

[GZ94] Gupta K. and Zhu X. (1994) Practical motion planning for many degrees of freedom: A novel approach within sequential framework. In *Proc. of IEEE Int. Conf. Robotics and Automation*, pages 2038–2043. San Diego, CA.

[HLM97] Hsu D., Latombe J., and Motwani R. (1997) Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Robotics and Automation*. Minneapolis, MN.

[HST94] Horsch T., Schwarz F., and Tolle H. (1994) Motion planning for many degrees of freedom - random reflections at c-space obstacles. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 3318–3323. San Diego, CA.

[Kav93] Kavraki L. (1993) Computation of configuration-space obstacles using the fast fourier transform. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 255–261. Atlanta, GA.

[Kav95] Kavraki L. E. (1995) *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford University.

[Kav97] Kavraki L. E. (1997) Geometry and the discovery of new ligands. In Laumond J.-P. and Overmars M. (eds) *Algorithms for Robotic Motion and Manipulation*, pages 435–448. A. K. Peters.

[KKKL94] Koga Y., Kondo K., Kuffner J., and Latombe J. (1994) Planning motions with intentions. *Computer Graphics (SIGGRAPH'94)* pages 395–408.

[KKL96] Kavraki L., Kolountzakis M., and Latombe J. (1996) Analysis of probabilistic roadmaps for path planning. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, pages 3020–3025.

[KL94a] Kavraki L. and Latombe J.-C. (1994) Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2138–2145. San Diego, CA.

[KL94b] Kavraki L. and Latombe J.-C. (1994) Randomized preprocessing of configuration space for path planning: Articulated robots. In *Proc. IEEE/RSJ/GI Int. Conf. Intelligent Robots and Systems*, pages 1764–1772. Germany.

[KLMR95] Kavraki L., Latombe J., Motwani R., and Raghavan P. (1995) Randomized query processing in robot motion planning. In *Proc. ACM Symp. on Theory of Computing*, pages 353–362.

[Kod87] Koditschek D. (1987) Exact robot navigation by means of potential functions: some topological considerations. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1–6.

[Kon91] Kondo K. (1991) Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration. *IEEE Tr. on Robotics and Automation* 7(3): 267–277.

[KŠLO96] Kavraki L., Švestka P., Latombe J., and Overmars M. (1996) Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. and Autom.* 12: 566–580.

[Lat91] Latombe J. (1991) *Robot Motion Planning*. Kluwer, Boston, MA.

[LL96] Lamiraux F. and Laumond J. (1996) On the expected complexity of random path planning. In *Int. Conf. on Robotics and Automation*. Minneapolis, MN.

[LMCG95] Lin M., Manocha D., Cohen J., and Gottschalk S. (1995) Collision detection: Algorithms and applications. In Goldberg *et al.* [GHLW95], pages 129–141.

[LP83] Lozano-Pérez T. (1983) Spatial planning: A configuration space approach. *IEEE Tr. on Computers* 32: 108–120.

[LPO91] Lozano-Pérez T. and O'Donnel P. (1991) Parallel robot motion planning. In *Proc. IEEE Int. Conf. Rob. and Automation*, pages 1000–1007. Sacramento, CA.

[LPW79] Lozano-Pérez T. and Wesley M. (1979) An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. of the AC* 22(10): 560–570.

[LTJ90] Laumond J.-P., Taix M., and Jacobs P. (1990) A motion planner for car-like robots based on a global/local approach. In *Proc. IEEE Internat. Workshop Intell. Robot Syst.*, pages 765–773.

[OŠ95] Overmars M. and Švestka P. (1995) A probabilistic learning approach to motion planning. In Goldberg *et al.* [GHLW95], pages 19–37.

[Ove92] Overmars M. (1992) A random approach to motion planning. Technical Report RUU-CS-92-32, Utrecht University, the Neverlands.

[OY82] Ó'Dúnlaing C. and Yap C. (1982) A retraction method for planning the motion of a dis. *J. of Algorithms* 6: 104–111.

[RC94] Rimon E. and Canny J. (1994) onstruction of c-space roadmaps from local sensory data, what should the sensors look for? In *Proc. IEEE Int. Conf. Robotics and Automatio*, pages 117–123. San Diego, CA.

[Rei79] Reif J. (1979) Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symp. on Found. of Comp. Sci.*, pages 421–427.

[RK92] Rimon E. and Koditschek D. (1992) Exact robot navigation using artificial potential functions. *IEEE Tr. on Rob. and Autom.* 8: 501–518.

[ŠO97] Švestka P. and Overmars M. (1997) Motion planning for car-like robots using a probabilistic learning approach. *Int. J. of Robotics Research* 16(2): 119–143.

[Šve97] Švestka P. (1997) *Robot Motion Planning using Probabilistic Road Maps*. PhD thesis, Utrecht University, the Netherlands.

—