Open access • Book Chapter • DOI:10.1007/978-1-4615-0013-1_8

A Randomized Approach to Robot Path Planning Based on Lazy Evaluation
— Source link ↗

Robert Bohlin, Lydia E. Kavraki

Related papers:

- Probabilistic roadmaps for path planning in high-dimensional configuration spaces

- Randomized query processing in robot path planning (Extended Abstract).

- Robot Path Planning Method Based on Level Set

- A* Algorithm Based Robot Path Planning Method

- On finding narrow passages with probabilistic roadmap planners

Chapter 1

A RANDOMIZED APPROACH TO ROBOT PATH PLANNING BASED ON LAZY EVALUATION

Robert Bohlin
Department of Mathematics
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

Lydia E. Kavraki
Department of Computer Science
Rice University
Houston, TX 77005, USA

Abstract   Path planning addresses the problem of finding collision-free paths for moving objects – robots – among obstacles. Randomized techniques have shown great performance in high-dimensional configuration spaces and are now the methods of choice for complex problems. In this paper we describe the Probabilistic Roadmap Method (PRM), variations of PRM, and other closely related algorithms. PRM is a simple and widely used planner applicable to virtually any kind of robot. The underlying idea is to build a roadmap in the configuration space. The nodes in the roadmap correspond to feasible configurations selected at random, and the edges correspond to feasible path segments. In the query phase, the initial and goal configurations are connected to the roadmap, and then the planner searches for a shortest path.

   The contribution of this paper is a new scheme for the lazy evaluation of the feasibility of the roadmap. We apply the scheme to PRM and obtain a planner called Lazy PRM. The overall theme of the algorithm is to increase speed by only exploring the part of the roadmap that is necessary for the current query. Lazy PRM is tailored to efficiently answer single planning queries, but can also be used for multiple queries. Experimental results provided in this paper show that our lazy method is very efficient in practice.

Keywords: Collision avoidance, motion planning, path planning, probabilistic roadmaps, randomized path planning, robotics.

1

1.   INTRODUCTION AND MOTIVATION

Path planning for robots is a broad and intensively studied problem. The general problem is to find collision-free paths for moving objects among a set of obstacles. The moving objects are called robots and they operate in an environment called the workspace. A robot can be a single rigid body or a collection of rigid bodies connected by joints.

Path Planning Applications. Planners are used in a wide variety of applications. Industrial robots weld, paint, and assemble products and their actions could be programmed automatically to a large extent. Many Automated Guided Vehicles (AGVs) plan their paths in real-time while they transport assembly parts between different stations in a workshop. AGVs use sensors to navigate and to detect unknown moving obstacles (like humans). Planners are integrated with Computer-Aided Design (CAD) software to support the design engineer in deciding whether or not it is possible to assemble or maintain a product.

More applications can be found in computer graphics where planners generate motions for animated humans, vehicles and other objects [26]. The task may also involve coordination of several robots. One important issue here is to generate motions that look natural and smooth. In medical surgery, planners are used to generate paths for operations so that the damage in tissues involved is minimized. Neurosurgeons, for example, use the planner in [40] to compute how to generate radiation beams that destroy brain tumors without damaging surrounding tissues. Last but not least, planning has interesting applications in biomedicine. For example, a path planner can be used to compute if a drug molecule (the robot) can find a path into a large protein molecule, dock to its active site, and in this way prevent an undesired reaction in the active site [15, 28].

Need for Efficient Path Planners. There are several reasons for using path planners in the above applications. In many industrial problems and in computer graphics animation, the main reason is to reduce programming time. Manual calculation of paths can be very tedious. For example, a car body may have thousands of spots to weld and there the potential time savings from the use of a planner are huge. The quality of the produced paths is also an important issue. Often, execution time is also critical, and planners can be used to optimize paths in that respect.

A planner may also allow the computation of very complex motions. An industrial example may be the coordination of fixture motion and robot motion in an arc welding application. In computer graphics, one

may want to have a large number of virtual characters interacting, e.g., a crowd of tourists can be made to follow a guide. A car-like robot has curvature constraints on the paths it can follow due to its limited turning radius. Constraints of a different type arise when a dextrous robot grasps and manipulates an object, or when cooperating robots simultaneously manipulate an object. Then closed kinematic chains appear and these must always be maintained, otherwise the object will fall down. Another difficulty that appears in real scenarios is dealing with uncertainties. We may have uncertainties in the control of the robot, in sensing, or incomplete knowledge of the environment (e.g., the Pathfinder rover that was sent to explore the surface of Mars [34]).

Our Work and Outline of the Paper. Different path planning algorithms have been developed for addressing the issues above. We concentrate on the most basic version of the path planning problem, that of moving a robot in a static environment. Efficient solutions of that problem translate to improvements in the solution of problems with more constraints such as the ones mentioned above. The high computational complexity of the basic path planning problem (see Section 2) dictates the use of randomized techniques for its solution.

In the first part of this paper, we revisit some general randomized path planning techniques for robots with many degrees of freedom (dof). The class of algorithms we are mostly interested in are the Probabilistic Roadmap Methods (PRMs). These algorithms are simple, general and can be used for a broad range of problems. In the second part of the paper, we present a new planner based on the PRM framework. The planner tries to minimize the number of collision checks done in the PRM framework and hence increase speed. We call the new planner Lazy PRM. We demonstrate that Lazy PRM is particularly effective in standard industrial applications. The latter are high-dimensional problems characterized by complex geometry and relatively uncluttered spaces.

This paper is organized as follows. Section 2 discusses the complexity of the basic path planning problem, motivating the need for randomized techniques for its solution. Section 3 establishes the terminology that will be used in the rest of the paper. Section 4 gives a description of the basic PRM, a number of variations of the algorithm, and other closely related algorithms. In Section 5 we describe Lazy PRM. We draw our discussion from [6, 7, 8]. Related ideas about lazy evaluation have been developed concurrently and independently in [35]. An analysis of the planner is given in Section 6 while experimental results are given in Section 7. We conclude in Section 8 with a discussion of the capabilities and limitations

of Lazy PRM. Some of our comments apply to randomized approaches to path planning in general.

2. COMPLEXITY ISSUES

In the previous section we gave some examples of path planning applications with different characteristics, and some problems that arise in practice. Unfortunately, there is no planning technique that deals with all of those conditions and constraints. For some special cases there exist good planners, but in general there is still a way to go until planners can be practically useful.

An algorithm is called *complete* if it always will find a solution or determine that none exists. Complete algorithms for the basic path planning problem exist, but they can not be used in practice due to their high complexity. So far, the fastest complete algorithm for arbitrary robots is given in [12] and it is exponential in the number of dof of the robot. The algorithm is too slow to be useful in practice, and it is mostly used in theoretical analysis as an upper bound on the complexity of the path planning problem. Some versions of the path planning problems have been proved PSPACE-hard [38]. Various other complexity results are described in [15, 27].

An alternative to complete planners are *probabilistically complete* planners. If a collision-free path exists, a probabilistically complete planner finds a solution with a probability approaching 1 given enough time for the computation [15]. Trading completeness for speed, randomized techniques have been successfully applied to generate path planners that are now the methods of choice for complex planning tasks. In this paper, we will explore in depth one such technique, the PRM framework, which has delivered excellent experimental results in the last five years.

3. NOTATION

We need a way to describe the position of the robot in a convenient way. For instance, if the robot is a single rigid object in a three-dimensional workspace, it has six dof and we can specify its position and orientation by six parameters; three coordinates for the translation, and three angles for the orientation. Similarly, the pose of an articulated robot arm (with a fixed base) is completely specified by the joint values.

More generally, a *configuration* is a set of independent parameters such that the position of every point of the robot can be determined relative to a fixed frame in the workspace. The set of all configurations is called the *configuration space* and is denoted by $\mathcal{C}$. The dimension of $\mathcal{C}$ is equal to the number of dof of the robot. The obstacles in the

workspace can be mapped onto a subset of $\mathcal{C}$ – the configuration space obstacles – by the following definition: a configuration belongs to the configuration space obstacles if the robot intersects any obstacle in the workspace. The open subset of collision-free configurations is denoted by $\mathcal{F}$.

A path for the robot is simply a continuous curve in $\mathcal{C}$. We will also refer to a path as a sequence of points in $\mathcal{C}$, in which case the path is the piecewise linear curve obtained by linearly interpolating subsequent points.

With this notation, we can rephrase the basic path planning problem as follows: given an initial configuration $\boldsymbol{q}_{init}$ and a goal configuration $\boldsymbol{q}_{goal}$ in $\mathcal{F}$, find a continuous curve in $\mathcal{F}$ connecting these points, or determine that none exists [27]. This formulation of the problem is usually favorable, since it is stated in terms of navigating a point rather than objects in the workspace. In that sense, the planning problem becomes independent of the geometry and kinematics of the robot, and may appear simpler. However, as soon as the robot is allowed to rotate or has revolute joints, the kinematics is non-linear, and seemingly simple obstacles in the workspace are mapped onto very complex obstacles in the configuration space. If the dimension of $\mathcal{C}$ is low, say four or less, then it is possible to obtain an approximate representation of the configuration space obstacles by discretization. But if the dimension is higher there is no convenient way to represent $\mathcal{C}$.

So, how can we plan in a space where the obstacles cannot be represented? We do have an implicit representation of the obstacles. Given a configuration it is generally easy to use the forward kinematics function to calculate the position of the robot in the workspace. Then we can test for intersection with the workspace obstacles to decide whether the configuration is feasible or not. Thus, we can point-wise determine if we are in $\mathcal{F}$. As a consequence, we cannot completely verify that a path, i.e. a curve in $\mathcal{C}$, is entirely collision-free, but we consider a path being feasible if it is collision-free to a certain resolution.

4.   RANDOM ROADMAPS FOR FAST PATH PLANNING

The idea of using randomization to solve the path planning problem is not new. The Randomized Path Planner (RPP) [5] was one of the first algorithms that used randomization. The algorithm was a breakthrough that made it possible to solve difficult problems in high-dimensional configuration spaces. At the same time, RPP possessed completeness properties. The planner uses a potential field in the configuration space

and calculates the potential without an explicit representation of the configuration space obstacles. The potential field is composed of one attractive field guiding the search towards the goal, and one repulsive field preventing from collisions with the obstacles. Starting from q_{init}, the algorithm follows the steepest descent direction towards q_{goal}. However, the potential is not perfect (calculating an ideal potential is probably as difficult as the path planning problem in itself) and generally contains local minima. To escape local minima the RPP combines descent motions and random walks.

One of the most general and widely used methods for path planning is the Probabilistic Roadmap Method (PRM). The algorithm is easy to implement and has been successful in many applications, also in high-dimensional configuration spaces. PRM is practically independent of the geometry and the kinematics of the robot, and can be used with virtually any kind of robot. In this section we will start with a detailed description of a basic PRM and continue with some variations of PRM.

4.1 THE PROBABILISTIC ROADMAP METHOD (PRM)

The idea behind the Probabilistic Roadmap Method (PRM), described in [24, 25, 37], is to represent and capture the connectivity of $\mathcal{F}$ by a random network – a *roadmap*. The nodes in the roadmap correspond to randomly selected configurations, and the edges correspond to path segments between the nodes. In a preprocessing step, or a *learning phase*, a large number of points are distributed uniformly at random in $\mathcal{C}$, and those found to be in $\mathcal{F}$ are retained as nodes in the roadmap. A local planner is then used to find paths between each pair of nodes that are sufficiently close together. If the planner succeeds in finding a path between two nodes, they are connected by an edge in the roadmap. In the *query phase*, the user specified start and goal configurations are connected to the roadmap by the local planner. Then the roadmap is searched for a shortest path between the given points, see Figure 1.1.

Even though a powerful local planner will require few nodes to obtain a well connected roadmap, most implemented PRMs show that it is computationally more efficient to distribute nodes densely and use a relatively weak, but fast, local planner, see [25, 37]. The local planner may for instance only check the straight line between two nodes. Other local planners are discussed and evaluated in [1].

Often the learning phase of PRM has a *node enhancement* step in order to increase the connectivity of the roadmap by adding more nodes in difficult regions of $\mathcal{F}$. Different techniques are used to identify these
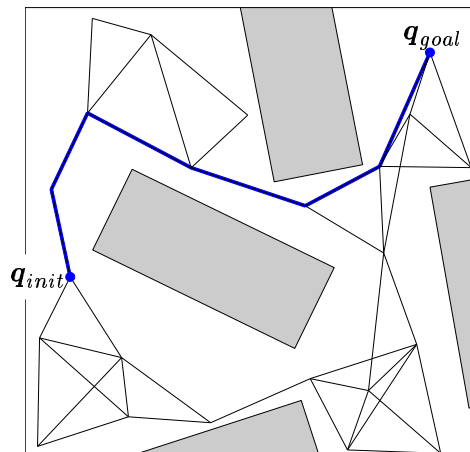
Figure 1.1   A random roadmap created by basic PRM in a simple two-dimensional configuration space with rectangular obstacles (grey). The shortest path is the thick line.

regions; one way is to distribute new points close to a number of *seeds* randomly selected among the existing nodes. In [24], the probability that a node is selected is proportional to $\frac{1}{1+b}$, where b is the number of edges connected to the node. An alternative selection can be based on a node's ratio of failed attempts by the local planner to find paths to other nodes [25]. Other techniques to increase the connectivity of the roadmap are described in [2] and [17].

PRM has shown to work well in practice in high-dimensional configuration spaces, see [25]. Indeed, it is useful for multiple queries, since once an adequate roadmap has been created, queries can be answered very quickly.

4.2   VARIATIONS OF PRM

The idea of using randomization, and the simplicity and generality of randomized algorithms, have inspired further development of PRM and other closely related methods. A few algorithms similar to PRM do not divide the planning process into a learning phase and a query phase. Given an initial and a goal configuration, the planner in [36] inserts randomly distributed nodes in $\mathcal{F}$, one at a time, and connects them to the different components of the roadmap by a local planner. New nodes are inserted until the initial and goal configurations can be found in the same connected component of the roadmap. See also [13] and [21]

for related algorithms. The latter paper gives an adaptive scheme for adjusting the power of the local planner.

Although the node enhancement step was developed to increase the connectivity of the roadmap, basic PRM still has weaknesses in finding paths through narrow passages in $\mathcal{F}$. Several recent approaches are intended to improve PRM in this respect by using different sampling strategies. The underlying idea is to distribute nodes close to the boundary of $\mathcal{F}$. The planner in [18] initially allows the robot to penetrate the obstacles to a certain extent. Small neighborhoods around the configurations just in collision are then re-sampled in order to place nodes close to the boundary of $\mathcal{F}$. The Obstacle Based PRM (OBPRM) in [2] and [3], repeatedly determines a configuration in collision to be the origin of a number of rays. Binary search is then used along each ray to find points on the boundary of $\mathcal{F}$, where roadmap nodes are placed. In [9], another idea is presented. The planner identifies the boundary of $\mathcal{F}$ by distributing points in pairs. Each pair is generated by first picking one point uniformly at random in $\mathcal{C}$, and then picking another point close to the first one. One of the points is added to the roadmap only if it is in $\mathcal{F}$ and the other point is not. Yet another technique to increase the number of nodes in narrow passages of $\mathcal{F}$ is presented in [41]. Points are picked uniformly at random in $\mathcal{C}$ and then retracted onto the medial axis of $\mathcal{F}$. Retraction to the medial axis of the workspace is done in [10].

Randomized approaches related to PRM are described in [19] and [30]. These build two trees rooted at the initial and goal configurations respectively. As soon as the trees intersect, a feasible path can be extracted. What differs these two methods is the way of expanding the trees. In [19], the trees are expanded by generating new nodes randomly in the vicinity of the two trees, and connecting them to the trees by a local planner. The planner in [30] iteratively generates a configuration, an attractor, uniformly at random in $\mathcal{C}$. Then, for both trees, the node closest to the attractor is selected and a local planner searches for a path of a certain maximum length towards the attractor. A new node is placed at the end of both paths. A new attractor is selected until the two trees intersect.

The algorithm in [29] is a method to keep the number of nodes in the roadmap to a minimum. Candidate nodes are generated uniformly at random, one at a time. A node is inserted to the roadmap only if it can be connected to at least two components of the roadmap, or if it does not see any other node. In the former case the components are merged, and in the latter case a new component is created. Variations of PRM have also been used for manipulation planning and for robots with closed kinematic chains, see [16, 31, 35].

5.   DESIGNING AN EFFICIENT RANDOMIZED PLANNER USING A LAZY EVALUATION SCHEME

5.1   MOTIVATION

In many applications, the configuration space changes frequently. For example, as soon as the robot changes tools, grasps or deforms an object, or when a new obstacle enters the workspace, the feasible part $\mathcal{F}$ is affected. A planner useful in practice must be able to plan in new configuration spaces instantly, so long preprocessing must be avoided. Ideally, the time required for planning should relate to the difficulty of the planning task, i.e., a simple path in an uncluttered environment should be found quickly, while a more complicated path may require more time.

In a similar way, the planning time should relate to the desired quality of the solution path. The quality of a path is difficult to quantify (see further discussion in Section 5.4), but in general we prefer short paths in $\mathcal{C}$, with respect to some metric.

We would also like the planner to learn to some extent, i.e., to use information from previous queries in order to speed up subsequent queries. For example, if the algorithm finds a path through a narrow passage in $\mathcal{F}$, it should be able to use that information when searching for a new path back through the passage.

The general theme for roadmap algorithms is to construct a network of paths verified to be collision-free by a local planner. Unfortunately, it is difficult to find a global strategy that can use these local planners efficiently in order to avoid traps and dead ends. In environments with complex geometry and expensive collision checks, this often means that too much time is spent on planning local paths that will not appear in the final path. So, even though PRM fulfills most of the above requirements, it is too slow in many applications.

Our solution is to avoid using local planners as much as possible, and instead keep a global view through the entire planning process. In this section we present Lazy PRM – a path planning algorithm tailored for single queries in high-dimensional, relatively uncluttered configuration spaces. We address the problem of finding simple paths quickly in industrial environments with complex geometry. In these environments collision checking is computationally expensive, so to make the planner fast, the main theme is to minimize the number of collision checks.

5.2   OVERALL SCHEME OF LAZY PRM

This section describes a new algorithm for single query path planning. The algorithm is similar to the basic PRM in [25] in the sense that the aim is to find the shortest path in a roadmap generated by randomly distributed configurations. In contrast with existing PRMs, we do not build a roadmap of feasible paths, but rather a roadmap of paths *assumed* to be feasible. The idea is to lazily evaluate the feasibility of the roadmap as planning queries are processed.

In other words, let q_{init}, q_{goal}, and a number of uniformly distributed configurations form nodes in a roadmap. We connect by edges each pair of nodes being sufficiently close together. Lazy PRM finds a shortest feasible path in the roadmap by repeatedly searching for a shortest path, and then checking whether it is collision-free or not. Each time a collision occurs, the corresponding node or edge is removed from the roadmap, and then Lazy PRM searches for a new shortest path.

This procedure can terminate in either of two ways. If there exist feasible paths in the roadmap between q_{init} and q_{goal}, we will find a shortest one among them. Otherwise, if there is no feasible path, we will eventually find q_{init} and q_{goal} in two disjoint components of the roadmap. In the latter case, we can either report failure, or, if we still have time, add more nodes to the roadmap in a similar way to the node enhancement in [24, 25], and start searching again. A high-level description of the algorithm is given in Figure 1.2.

The point by using this scheme for lazy evaluation is that we only explore the part of the roadmap that is needed for the current query. The scheme is simple, general and can be applied also to other roadmap planners in order to increase performance. The strength is to either find a collision-free path or to conclude that none exists in the roadmap by using a small number of collision checks. It is always an advantage to use lazy evaluation since we can never do more work, in terms of collision checking, than basic PRM would do.

The rest of this section explains the different steps of the algorithm in more detail, Section 6 gives a proof of its probabilistic completeness, and Section 7 shows some experimental results.

5.3   BUILDING THE INITIAL ROADMAP

The first step in the algorithm is to build a roadmap $\mathcal{G}$ in $\mathcal{C}$. There are two parameters that determine the size of $\mathcal{G}$; the number of nodes, N_{init}, and the expected number of neighbors, M_{neighb}, connected to each node.
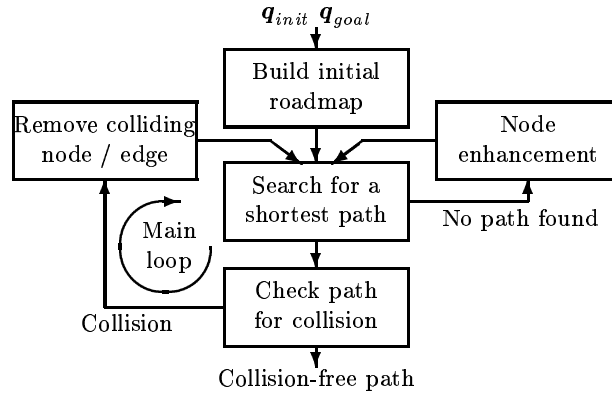
Figure 1.2   High-level description of Lazy PRM.

Initial Distribution of Nodes.   Initially, we distribute N_{init} points uniformly at random in $\mathcal{C}$. These points, together with $\boldsymbol{q}_{init} \in \mathcal{F}$ and $\boldsymbol{q}_{goal} \in \mathcal{F}$, form nodes in $\mathcal{G}$. An important issue is the choice of N_{init}. The initial density of nodes, determined by N_{init}, is strongly correlated to the probability of finding a short path, if one exists. The correlation is hard to quantify, but the following example may give an illustration. Assume there exist only two ways to get to the goal configuration; either a short path through a rather narrow corridor, or a somewhat longer path through a wide corridor. If $\mathcal{G}$ is sufficiently dense, the algorithm will find a short path through the narrow passage, see Figure 1.3(b). If $\mathcal{G}$ is sparse, the algorithm will find a longer path through the wide passage, see Figure 1.3(a). In the worst case, if the roadmap is too sparse, there will be no feasible path at all in the roadmap, and the algorithm has to go to the enhancement step to generate more nodes. On the other hand, if N_{init} is too large, we will distribute more nodes than necessary, see Figure 1.3(c). Although we may obtain better paths, this will lead to somewhat longer planning times.

However, the idea behind the algorithm is that only a small fraction of the nodes in the roadmap will be necessary to check for collision. This makes the algorithm relatively insensitive to high density of nodes, so we can choose N_{init} relatively large. (In our experiments we start with $N_{init} = 10000$ nodes and check on average 322 nodes in the most difficult planning task, see Task $D \rightarrow E$ in Table 1.1(a), Section 7.) The number of nodes required to find a path is further explored in Section 6.

Selecting Neighbors.   To build the roadmap we connect each node in $\mathcal{G}$ by edges to a set of neighbor nodes. An edge represents the straight
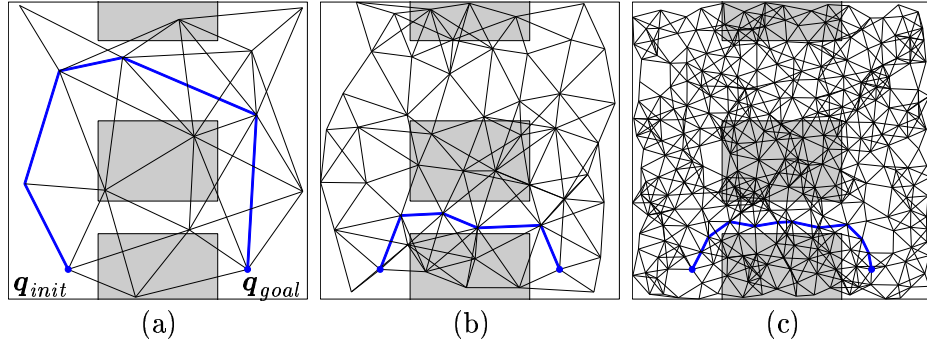
Figure 1.3 Example of a two-dimensional configuration space with rectangular obstacles (grey). The thick lines show the shortest feasible paths between q_{init} and q_{goal} in three different roadmaps. The roadmap in (a) is too sparse and no short feasible path exists. The roadmap in (c) is very dense, and the shortest feasible path will take longer time to find than the shortest feasible path in (b).

line path in $\mathcal{C}$ between two nodes. Neither the nodes nor the edges are being checked for collision in the initial step, but we want, of course, to have edges which are likely to be feasible. Since it would require far too much memory to connect all pairs of nodes, and it is unlikely that the straight line path between two nodes far apart is feasible, it is natural to only consider nodes which are sufficiently close together.

In order to select appropriate neighbors, we need a metric $\rho_{coll} : \mathcal{C} \times \mathcal{C} \to [0, \infty)$ such that the distance between two configurations under this metric reflects the difficulty of connecting them by a collision-free straight line path. Then we connect each pair of nodes (q, q') such that $\rho_{coll}(q, q') \leq R_{neighb}$. For any fixed radius R_{neighb}, the number of neighbors of a node is a random variable, so depending on the initial number of nodes N_{init}, we choose R_{neighb} such that the expected number of neighbors equals the parameter M_{neighb} introduced in the beginning of Section 5.3.

In many cases it is harder to make feasible connections in certain directions than in others. Consider for instance an articulated robot arm; then it is more likely that a collision occurs when the base joint is moving one unit, than if a joint close to the end-effector is moving one unit. With this in mind, we let ρ_{coll} be a weighted Euclidean metric,

$$
\begin{aligned}
\rho_{coll}(\boldsymbol{x}, \boldsymbol{y}) &= \Big(\sum_{i=1}^{d} w_i^2 (x_i - y_i)^2 \Big)^{1/2} \\
&= \big((\boldsymbol{x} - \boldsymbol{y})^T W (\boldsymbol{x} - \boldsymbol{y}) \big)^{1/2}, \qquad (1.1)
\end{aligned}
$$

where d is the dimension of $\mathcal{C}$, $\{w_i\}_{i=1}^d$ are positive weights, $W = diag(w_1^2, \ldots, w_d^2)$, and $\boldsymbol{x}^T$ is the transpose of $\boldsymbol{x}$. The weights are chosen in proportion to the maximum possible distance (Euclidean distance in the workspace) traveled by any point on the robot, when moving one unit in $\mathcal{C}$ along the corresponding axis. This metric is easy to use and has been shown to work well in our experiments presented in Section 7.

5.4   SEARCHING FOR A SHORTEST PATH

The second step in the algorithm is to find a shortest path in $\mathcal{G}$ between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, or determine that none exists. We use the A^* algorithm [33], and a metric $\rho_{path} : \mathcal{C} \times \mathcal{C} \to [0, \infty)$ to measure the length of a path and the remaining distance to $\boldsymbol{q}_{goal}$.

If the search procedure succeeds in finding a path, we need to check it for collision. Otherwise, if no path exists in the roadmap, we either report failure, or go to the node enhancement step to add more nodes to the roadmap and start searching again depending on the overall time allowed to solve the problem.

Choosing an Appropriate Metric for A^*. The tool available to give preference to certain paths and reject others is the metric ρ_{path}. Thus, by defining this metric we decide which paths are assumed to be of high quality and which paths are assumed to be of poor quality.

In this paper we focus on articulated robots and use the Euclidean configuration space $I_1 \times \cdots \times I_d$, where I_i is the range of joint i and d is the number of dof. Thus, we do not identify angles equal modulo 2π as being equal, although they define the same position in the workspace. This is because a real robot in general has supply wires, etc., which otherwise would be entangled. The metric ρ_{path} is a weighted Euclidean metric, similar to (1.1), where the weights are equal to $\frac{1}{v_i}$, $i = 1, ..., d$, where v_i is the maximum angular velocity of joint i. This tends to give preference to paths with short execution time, which in many applications is the most interesting response variable.

In the general case, however, there are a large number of other response variables to consider. Some of them are measurable such as energy consumption, dynamic forces on joints, etc. Others are more subjective; for example, the motion should look natural and smooth from the user's point of view. Under any Euclidean metric, the straight line path in $\mathcal{C}$ between two configurations is the shortest, but considering all of these response variables, the straight line path is not necessarily optimal. Thus, the choice of a configuration space parameterization and an appropriate metric is an intricate task in itself.

5.5   CHECKING PATHS FOR COLLISION

When the A^* algorithm has found a shortest path in the roadmap between q_{init} and q_{goal}, we need to check the nodes and edges along the path for collision. In most applications it is straightforward to perform a collision check for a given configuration, i.e. determine whether a point is in $\mathcal{F}$ or not [11, 32, 39]. It is considerably more expensive to check whether a path segment is entirely in $\mathcal{F}$ or not. To keep the planner as simple and general as possible, we only use a collision checker for points in $\mathcal{C}$; path segments, i.e. edges in the roadmap, are discretized and checked with a certain resolution. However, in [6, 7] we describe a variation of the algorithm which makes use of a function giving the minimum distance between the robot and the obstacles.

The overall purpose of the Search, Check, and Remove steps of our algorithm (the main loop in Figure 1.2), is roughly to identify and remove colliding nodes and edges from the roadmap until the shortest path between q_{init} and q_{goal} is feasible. Accordingly, when checking a path for collision, we are not primarily interested in verifying whether an individual node or edge is in $\mathcal{F}$ or not, but rather to remove colliding nodes and edges as efficiently as possible. Since a removal of a node implies all its connected edges to be removed, it seems reasonable to check the feasibility of the nodes along the path before checking the edges.

Checking Nodes.   Starting respectively with the first and the last node on the examined path and working toward the center, we alternately check the nodes along the path. As soon as a collision is found, we remove the corresponding node and its connected edges from the roadmap, and search for a new shortest path.

The reason for checking the nodes in this order is that the probability of having the shortest feasible path via a particular node is higher if the node is close to either q_{init} or q_{goal}. Consider, for instance, the nodes connected to q_{init}; a shortest feasible path (if one exists) must pass through at least one of them. Since, in a cluttered space, we cannot give preference to certain directions, the probability of having the shortest feasible path via a particular neighbor of q_{init} is at least $1/b$, where b is the number of neighbors of q_{init}. Nodes connected to q_{goal} have a similar probability, whereas nodes further away from both q_{init} and q_{goal} have a much lower probability of being in the shortest feasible path. Therefore, we check the nodes along a path starting from the end-nodes and working toward the center.

Checking Edges.   If all nodes along the path are in $\mathcal{F}$, we start checking the edges in a similar fashion; working from the outside in. However,

to minimize the risk of doing unnecessary collision checks, we first check all edges along the path with a coarse resolution, and then do stepwise refinements until the specified resolution is reached. As with the nodes, if a collision is found, we remove the corresponding edge, and search for a new shortest path. If no collision is found along the path, the algorithm terminates and returns the collision-free path. Figure 1.4 gives an illustration. To make the overall algorithm efficient, we record which nodes have been checked for collision, and to which resolution each edge has been checked, in order to avoid checking any point in $\mathcal{C}$ more than once.

The total number of collision checks depends on the resolution with which the edges along the path are checked. Again, since ρ_{coll} reflects the probability of collision, we determine the resolution with respect to this metric. The resolution is quantified by a step-size δ, but we prefer not to let the user specify the step-size by a certain number, because the resolution should depend on the scale of $\mathcal{C}$ and the weights defining the metric. A better way is to introduce a parameter M_{coll}, specifying the number of collision checks required to check the longest possible straight line path in $\mathcal{C}$. In other words, assuming that $\mathcal{C}$ is a d-dimensional rectangle and $\boldsymbol{q}$ and $\boldsymbol{q}'$ are two opposite corners, the step-size is related to the length of the diagonal of $\mathcal{C}$ according to

$$\delta = \frac{\rho_{coll}(\boldsymbol{q}, \boldsymbol{q}')}{M_{coll}}.$$

5.6 NODE ENHANCEMENT

If the search procedure in Figure 1.2 fails, no feasible path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ exists in the roadmap, and more nodes are necessary in order to find one. In the node enhancement step, we generate N_{enh} new nodes, add them to $\mathcal{G}$, and select neighbors in the same way as when $\mathcal{G}$ was initially built.

We may not only distribute the new nodes uniformly, but rather use the information available in the roadmap (or what is left of the roadmap), in order to distribute new nodes in difficult regions of $\mathcal{C}$. In a method similar to the node enhancement in [24, 25], we randomly select a number of points in $\mathcal{G}$, called *seeds*, and then distribute a new point close to each of them. Our experience is that it is better to select many seeds and distribute one new node around each of them, instead of selecting few seeds and distribute several nodes around each of them; the latter method is more dependent on the selection of seeds.

Although the seeds may help us identify difficult regions of $\mathcal{C}$, we still want to maintain a smooth distribution all over $\mathcal{C}$, because the knowledge
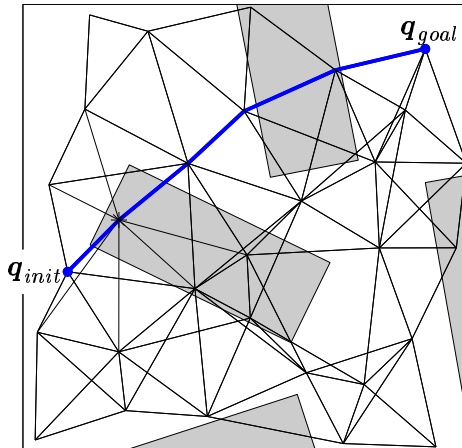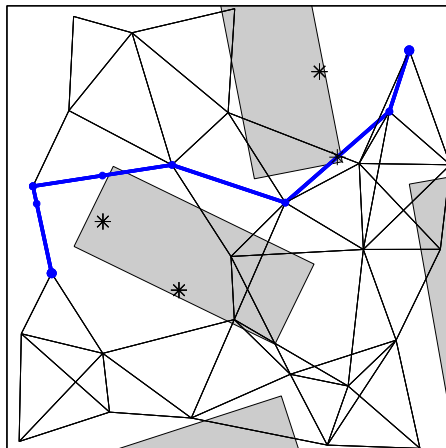
(a): Lazy PRM searches for a shortest path and checks the nodes. A collision is detected ($*$) and corresponding node is removed from the roadmap.
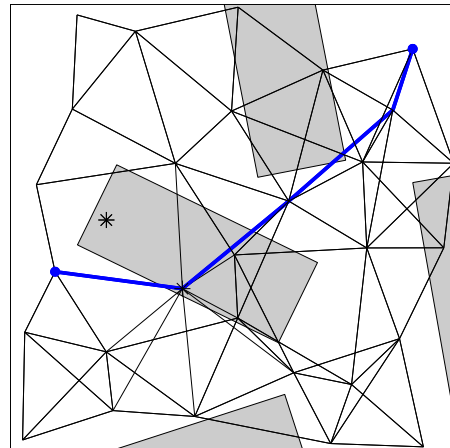
(b): Then Lazy PRM searches for a new shortest path, detects a new collision ($*$) and removes corresponding node.

(c): After a few iterations, a sequence of feasible nodes is found. When checking the edges with a coarse resolution a collision is found ($*$). The edge is removed from the roadmap, and the planner searches for a new shortest path.
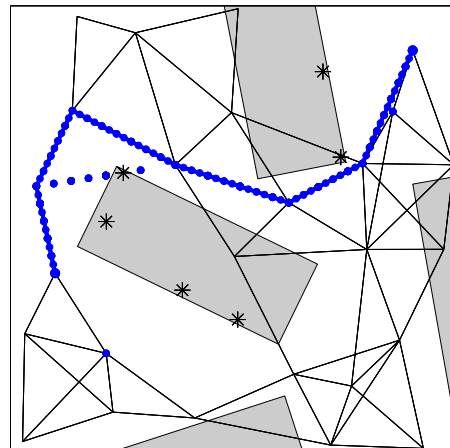
(d): Eventually, the planner finds a path whose nodes are collision-free, and whose edges are collision-free to a specified resolution.

Figure 1.4 Example of a planning query in a two-dimensional configuration space with rectangular obstacles (grey). *All* collision checks performed are marked with $*$ (collision) or $\bullet$ (collision-free).

about $\mathcal{C}$ is limited and we do not want to rely too much on the selection of seeds. To ensure probabilistic completeness (see Section 6), we also distribute new nodes uniformly at random in each step. In our algorithm, we let half of the enhancement nodes be uniformly distributed, and the rest distributed around seeds.

Selecting Seeds.   The set of edges which have been removed from the roadmap and have at least one end-point in $\mathcal{F}$ will certainly intersect the boundary of $\mathcal{F}$. Using the mid-points of these edges as seeds may help us distribute points close to the boundary of $\mathcal{F}$, thus increase the probability of finding paths through narrow passages in $\mathcal{F}$.

However, if the enhancement step is executed several times, this may cause problems with clustering of nodes. Assume that we add a new node q. This node will give rise to a number of edges which in the next enhancement step may increase the probability of adding even more nodes close to q. Thus, the distribution of new enhancement nodes depends on the preceding enhancement steps, and may eventually cause undesired clusters of nodes. To avoid this phenomenon, we only use edges whose end-nodes are generated uniformly at random when selecting seeds.

Distributing New Nodes.   When distributing a new point q around a seed η, we use the multivariate normal distribution. This distribution is smooth, easy to use, and allows us to control the distribution of q in terms of the metric ρ_{coll}. Hence, we can stretch the distribution in directions where the probabilities of making feasible connections are higher.

Introducing two parameters $\alpha \in (0, 1)$ and $\lambda > 0$, we can choose the distribution such that

$$\rho_{coll}(\boldsymbol{q}, \boldsymbol{\eta}) \leq \lambda R_{neighb} \tag{1.2}$$

is an event with probability $1-\alpha$, see Figure 1.5. R_{neighb} is the maximum length of an edge defined in Section 5.3. To achieve this property, we define a covariance matrix Σ as follows:

$$\Sigma = \frac{\lambda^2 R_{neighb}^2}{\chi_d^2(\alpha)} W^{-1}. \tag{1.3}$$

Here W is the same as in (1.1) and $\chi_d^2(\alpha)$ is the upper α percentile of a χ^2-distribution with d dof. Then we let the new point $\boldsymbol{q} \sim N_d(\boldsymbol{\eta}, \Sigma)$, i.e., $\boldsymbol{q}$ is multivariate normally distributed with d dof, mean $\boldsymbol{\eta}$, and covariance matrix Σ. Since Σ is diagonal, this simply means that each component $q_i, i = 1, \ldots, d$, of $\boldsymbol{q}$ is normally distributed with mean η_i and variance $\Sigma_{i,i}$.
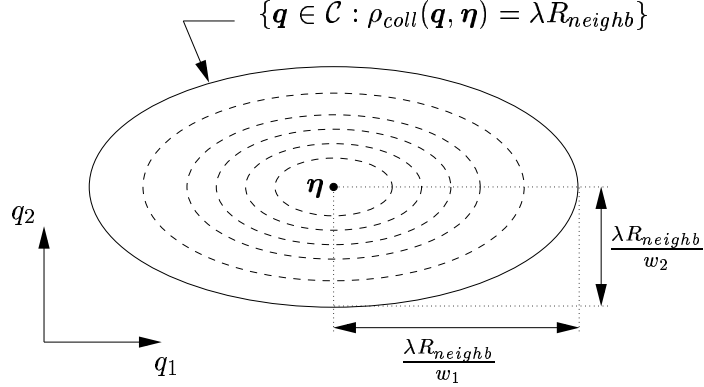
Figure 1.5 Example of a seed $\boldsymbol{\eta}$ in a two-dimensional configuration space. If a new point $\boldsymbol{q}$ is distributed according to $N_d(\boldsymbol{\eta}, \Sigma)$, with Σ as in (1.3), then $\boldsymbol{q}$ is distributed within the confidence ellipse (solid line) with probability $1 - \alpha$. The dashed ellipses are contours of the distribution function. w_1 and w_2 are the weights defined in (1.1).

To show (1.2), we use that $(\boldsymbol{q} - \boldsymbol{\eta})^T \Sigma^{-1} (\boldsymbol{q} - \boldsymbol{\eta})$ is χ^2-distributed with d dof [22]. Thus, the event

$$(\boldsymbol{q} - \boldsymbol{\eta})^T \Sigma^{-1} (\boldsymbol{q} - \boldsymbol{\eta}) \leq \chi^2_d(\alpha)$$

has probability $1 - \alpha$. Using (1.1) and (1.3) gives the confidence ellipsoid in (1.2).

We see in (1.3) that Σ depends on the the ratio $\lambda^2 / \chi^2_d(\alpha)$. Since both λ^2, $\lambda > 0$, and $\chi^2_d(\alpha)$, $\alpha \in (0, 1)$, are continuous functions whose ranges are $(0, \infty)$, one of the two parameters α and λ is redundant, so we can without loss of generality choose $\alpha = 0.05$. Then, the parameter λ controls the size of the 95% confidence ellipsoid relative to R_{neighb} as shown in Figure 1.5. In our experiments we found that $\lambda = 1$ is a suitable choice.

Another possibility of distributing the new point $\boldsymbol{q}$, is to let it be uniformly distributed in a rectangular box centered at $\boldsymbol{\eta}$. If we let the sides of the box be of equal length under ρ_{coll}, we stretch the box in a similar way as the ellipsoids above. In our path planning algorithm, however, the normal distribution has a major advantage compared to the uniform distribution; the contours of the distribution function are ellipsoids around $\boldsymbol{\eta}$ (see Figure 1.5). Hence, under the metric ρ_{coll}, which reflects the difficulty of making connections, the distribution is symmetric around $\boldsymbol{\eta}$. In contrast, the uniform distribution favors the directions

of the corners of the box, and nodes are more frequently distributed there than in other directions.

5.7   MULTIPLE QUERIES

When the planner has found a collision-free path, it terminates and returns the path. The information about which nodes and edges have been checked for collision is stored in the roadmap. As long as the configuration space remains the same, we use the same roadmap when processing subsequent queries. Thus, we benefit from the information obtained in each planning query. The new initial and goal configurations are simply added to the roadmap, and the same algorithm, except for the initial generation of nodes, is run again.

As several queries are processed, more and more of the roadmap will be explored, and the planner will eventually find paths via nodes and edges which have already been checked for collision. This makes the planner efficient for multiple queries.

Even in the long run, many nodes and edges may never be explored since they are located in odd regions of $\mathcal{C}$. Thus, given a fixed size of the roadmap, the number of collision checks performed by Lazy PRM will never exceed the number of collision checks performed by the basic PRM described in Section 4.1. Accordingly, there is no reason to entirely evaluate the roadmap unless we explicitly want it. The lazy evaluation scheme will find the shortest feasible path in the roadmap by using less collision checks.

6.   PROBABILISTIC COMPLETENESS

In this section we give a proof of probabilistic completeness of Lazy PRM. Before stating the theorem, which also can be found in [7], we need some notation. Let $\gamma : [0, L] \to \mathcal{F}$ be a curve (also called path) parameterized by arc length and with continuous tangent. A *tube* τ of radius r around $\gamma(s)$ is the set of points at distance r from γ measured perpendicular to the tangent $\gamma'(s)$. Similarly, the corresponding *solid tube* is the set of points at distance $\leq r$ from γ. For simplicity, we usually omit the word solid.

A *regular tube* is a tube that does not intersect itself. If γ is enclosed by a regular tube of radius r, this particularly implies that its curvature, $\kappa(s) = |\gamma''(s)|$, is bounded from above by $1/r$. Otherwise the tube would be folded. The following lemma, proved in [14], states a useful property of regular tubes.

Lemma 1 *The volume enclosed by a regular tube around a curve in a d-dimensional Euclidean space is the product of the length of the curve and the $(d-1)$-dimensional area of a cross-section.*

In other words, if B_r^d is the ball of radius r in a d-dimensional space, and μ_d the Lebesgue measure, we can express the volume of a regular tube τ of radius r around γ as

$$\mu_d(\tau) = L\mu_{d-1}(B_r^{d-1}) = Lr^{d-1}\mu_{d-1}(B_1^{d-1}), \qquad (1.4)$$

where L is the length of γ.

Assuming there exists a path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, enclosed by a regular tube in $\mathcal{F}$, the following theorem gives an upper bound on the probability of failure to find a path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$. The assumption of an enclosing tube in $\mathcal{F}$ is relevant since $\mathcal{F}$ is an open subset of $\mathcal{C}$. Moreover, the theorem says that the probability of failure decreases exponentially in the *total* number of uniformly distributed nodes N. Since N increases in each enhancement step (Figure 1.2 and Section 5.6), the probability of failure vanishes as time tends to infinity. This is equivalent to the definition of *probabilistic completeness*, see [20]. Thus, Lazy PRM is a probabilistically complete path planner. Since the configuration space is at least two-dimensional (otherwise path planning is trivial), we assume that $d \geq 2$. Recalling the parameter R_{neighb} from Section 5.3 and the matrix W defined in (1.1) with norm $\|W\|$, we formulate the theorem as follows.

Theorem 1 *Let N be the total number of nodes generated uniformly at random in $\mathcal{C}$. If there exists a path γ between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, enclosed by a regular tube τ of radius $R \leq \frac{1}{2\|W\|^{1/2}}R_{neighb}$ entirely in $\mathcal{F}$, then Lazy PRM will fail to find a path with probability at most*

$$\frac{Ld}{R}e^{-\beta N},$$

where $\beta = \frac{R^d\,\mu_{d-1}(B_1^{d-1})}{3d\,\mu_d(\mathcal{C})}$ and L is the length of γ.

Proof. Let $u = R/d$, $r = R(1 - 1/d)$, and $k = \lfloor L/u \rfloor$. The idea of the proof is to take a tube of radius r, divide it into $k - 1$ cells of length u, and calculate the probability of having at least one node in each cell. We will show that any two points in adjacent cells can be connected by a straight line, and that one node in each cell is enough for the planner to succeed. Assume first that $k \geq 2$. The case $k < 2$ is trivial and will be considered at the end of the proof.
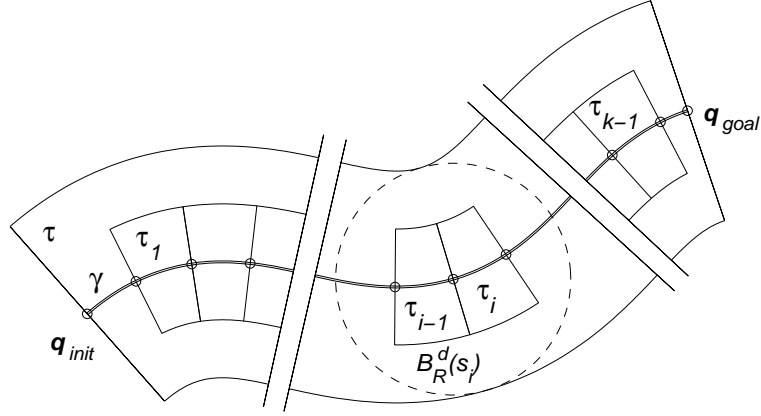
Figure 1.6 Illustration to the proof of Theorem 1.

Let $s_i = iu$, $i = 1, \ldots, k$, and let τ_i be the tube segment around $\gamma(s)$ for $s \in [s_i, s_{i+1})$, $i = 1, \ldots, k-1$, see Figure 1.6. The tube segments $\{\tau_i\}_{i=1}^{k-1}$ are pairwise disjoint and, by (1.4),

$$\frac{\mu_d(\tau_i)}{\mu_d(\mathcal{C})} = ur^{d-1} \frac{\mu_{d-1}(B_1^{d-1})}{\mu_d(\mathcal{C})}.$$

Now, for $d \geq 2$, $(1 - 1/d)^{d-1}$ is a decreasing function whose limit is e^{-1}, and since

$$ur^{d-1} = \frac{R^d}{d}\Big(1 - \frac{1}{d}\Big)^{d-1} \geq \frac{R^d}{d}e^{-1} \geq \frac{R^d}{3d},$$

we get that

$$\frac{\mu_d(\tau_i)}{\mu_d(\mathcal{C})} \geq \frac{R^d \, \mu_{d-1}(B_1^{d-1})}{3d \, \mu_d(\mathcal{C})} = \beta. \tag{1.5}$$

The N points generated by the algorithm are uniformly and independently distributed in $\mathcal{C}$. Thus, the probability that τ_i is empty equals $\big(1 - \frac{\mu_d(\tau_i)}{\mu_d(\mathcal{C})}\big)^N$, which, by (1.5), can be estimated:

$$\Big(1 - \frac{\mu_d(\tau_i)}{\mu_d(\mathcal{C})}\Big)^N \leq (1 - \beta)^N. \tag{1.6}$$

Let $B_R^d(s)$ be a ball of radius R centered at $\gamma(s)$, i.e., $B_R^d(s)$ has the same radius as τ. Unless $B_R^d(s)$ is close to the end-points of γ, it will be covered by τ, see Figure 1.6. If it is close to the end-points, however, it

might intersect the circular discs at the ends of the tube. Nevertheless, the intersection between $B_R^d(s)$ and τ is still convex, a property we will need later.

Now, let $\boldsymbol{q}_{i-1} \in \tau_{i-1}$ and $\boldsymbol{q}_i \in \tau_i$. By the definition of a tube there exists an $\sigma_i \in [s_i, s_{i+1})$ such that $|\boldsymbol{q}_i - \gamma(\sigma_i)| \leq r$. Since γ is parameterized by arc length, it follows that $|\gamma(s) - \gamma(t)| \leq |s - t|$, and, by the triangle inequality,

$$
\begin{aligned}
|\boldsymbol{q}_i - \gamma(s_i)| &\leq |\boldsymbol{q}_i - \gamma(\sigma_i)| + |\gamma(\sigma_i) - \gamma(s_i)| \\
&\leq r + u = R.
\end{aligned}
$$

Hence, the ball $B_R^d(s_i)$ contains τ_i. Similarly, we can show that it also contains τ_{i-1}. Since both cells are covered by τ, they are contained in the convex set $B_R^d(s_i) \cap \tau$ which is entirely in $\mathcal{F}$. Thus, $\boldsymbol{q}_{i-1}$ and $\boldsymbol{q}_i$ are at most $2R$ apart and the straight line between them lies entirely in $\mathcal{F}$. From (1.1) we get that

$$
\begin{aligned}
\rho_{coll}(\boldsymbol{q}_{i-1}, \boldsymbol{q}_i) &\leq |\boldsymbol{q}_{i-1} - \boldsymbol{q}_i| \, \|W\|^{1/2} \\
&\leq 2R \, \|W\|^{1/2} \\
&\leq R_{neighb},
\end{aligned}
$$

i.e., any node in τ_{i-1} is in the neighborhood of any node in τ_i and will therefore be interconnected by Lazy PRM. Moreover, since $\boldsymbol{q}_{init} \in B_R^d(s_1)$ and $\boldsymbol{q}_{goal} \in B_R^d(s_k)$, they will be connected to any node in τ_1 and τ_{k-1} respectively. Consequently, it is enough to have at least one node in each of the cells $\tau_1, \ldots, \tau_{k-1}$, in order for Lazy PRM to find a collision-free path between $\boldsymbol{q}_{init}$ or $\boldsymbol{q}_{goal}$.

The probability of failure for our algorithm, $P_{failure}$, can now be estimated:

$$
\begin{aligned}
P_{failure} &\leq P(\text{some } \tau_i \text{ is empty}) \\
&\leq \sum_{i=1}^{k-1} P(\tau_i \text{ is empty}) \\
&\leq (k-1)(1-\beta)^N,
\end{aligned}
$$

where we used Boole's inequality and (1.6) in the second and third step respectively. Using that $k - 1 \leq Ld/R$ and $(1-\beta)^N \leq e^{-\beta N}$ gives the desired estimation.

What remains is the case $k < 2$, i.e., $L < 2u = 2R/d \leq 2R$. Then both $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ are contained in the convex set $B_R^d(L/2) \cap \tau$ which is entirely in $\mathcal{F}$. This guarantees that Lazy PRM will find the straight

line path between q_{init} and q_{goal}, so the probability of failure is zero. $\square$

Note that a related theorem regarding basic PRM can be found in [4] and [23]. Both theorems give a bound on the failure probability expressed in terms of, among other variables, the density of nodes. An important difference is that Lazy PRM has to reach a certain density of nodes in $\mathcal{C}$, while basic PRM has to reach approximately the same density in $\mathcal{F}$. This seems like a weakness of our method, but looking at how the nodes in basic PRM are generated, we see that this is not the case. In order to reach the desired density in $\mathcal{F}$, basic PRM has to distribute nodes uniformly all over $\mathcal{C}$ and exclude those in collision. Consequently, for both algorithms to reach the same density, the number of nodes checked for collision in the learning phase of basic PRM has to be the same as the number of uniformly distributed nodes in Lazy PRM. So whether the density is specified in $\mathcal{F}$ or in $\mathcal{C}$ does not matter. The difference of practical significance is that Lazy PRM avoids checking *all* of the nodes for collision.

7.   EXPERIMENTAL RESULTS

In this section we present performance tests of Lazy PRM when applied to a six dof robot in a realistic industrial environment. The planner has been implemented in C++ as a plug-in module to RobotStudio[1] – a simulation and off-line programming software running under Windows NT. The collision checks are handled internally in RobotStudio. The experiments have been run on a PC with a 400 MHz Pentium II processor and 512 MB RAM. In all tests we let $N_{init} = 10000$, $M_{neighb} = 60$, $M_{coll} = 200$, and $N_{enh} = 500$.

7.1   PATH PLANNING TASKS

The test example is a part of a real manufacturing process in which an ABB 4400 robot is tending press breaking. Metal sheets are formed by the hydraulic press shown in Figure 1.7. In this particular example, plane sheets of metal are picked at a pallet, bent once at the hydraulic press, and then placed at another pallet.

The process is divided into several steps, and our aim is to automatically plan the unconstrained paths of the robot. We let A to E denote five different configurations shown in Figures 1.7 and 1.8. These are used as either initial or goal configurations in four planning tasks, denoted for

---

[1]RobotStudio is developed by ABB Robotics, Göteborg, Sweden.
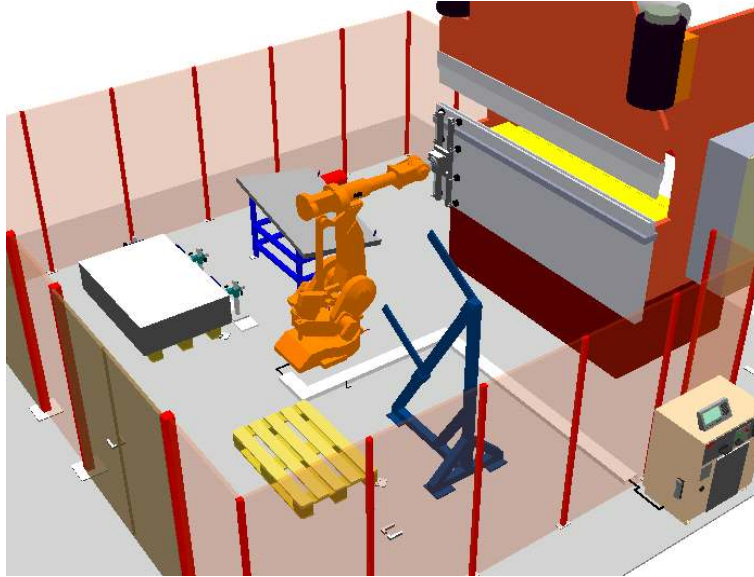
Figure 1.7 The work cell used in the experiments. The robot is in its home configuration denoted by A.

example $A \to B$, where A is the initial configuration and B is the goal configuration.

The scenario is as follows. Starting from the home configuration A, the robot picks a sheet of metal from the pallet at B (task $A \to B$) and puts the sheet-metal at the press C (task $B \to C$). After the breaking, the robot grasps the sheet-metal at D, places the sheet-metal at the pallet E (task $D \to E$), and then returns to the home configuration A (task $E \to A$).

Note that during this series of steps, the configuration space changes several times. As soon as we grasp or place a sheet of metal, the collision-free part, $\mathcal{F}$, is changing. Accordingly, we have four different configuration spaces in which to plan, and we have to build one roadmap in each of them.

The results include the number of collision checks, the number of enhancement steps, and the planning time. The minimum, average, and maximum values, based on 20 consecutive runs for each task, are shown in Table 1.1(a). The average number of collision checks performed on nodes and edges respectively are presented, as well as the average number of collision checks performed on the collision-free paths that the planner returned. Since paths are checked for collision with a certain resolution
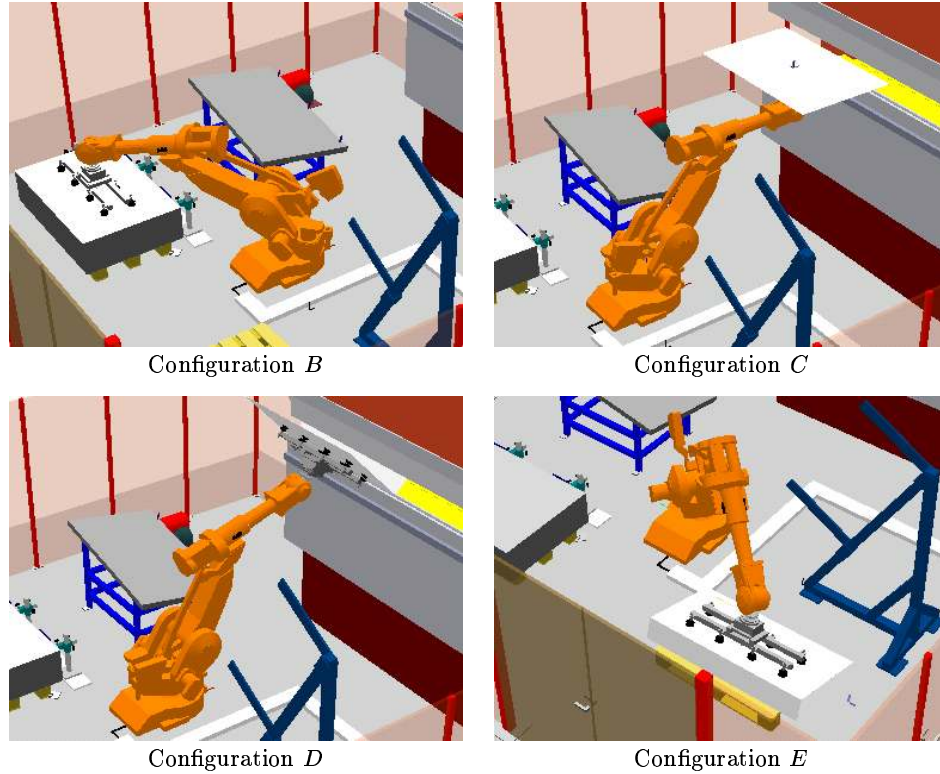
Configuration *B*

Configuration *C*

Configuration *D*

Configuration *E*

Figure 1.8   Configurations *B* to *E* used in the experiments.

(see Section 5.5), the latter figures correspond to the lengths of the collision-free paths.

The running times in Table 1.1(a) are divided into three parts. First, graph building, which includes distance calculations between nodes in $\mathcal{C}$ and node and edge adding, second, graph searching, and finally collision checking.

In the last column of Table 1.1, the average values of the recorded data are summed up. Thus, the last column indicates the average number of collision checks and average planning times for the entire press breaking operation.

In Table 1.1(b), we have included some results corresponding to the learning phase without node enhancement of basic PRM. For each task, we generated a roadmap in exactly the same way as Lazy PRM generates roadmaps in the initial step. Then we checked *all* nodes for collision,

deleted the colliding ones, and then checked *all* of the remaining edges as described in Section 5.5. In other words, we checked the *entire* roadmap for collision as efficiently as possible. Due to the long running times, only one full roadmap was explored for each task. The result gives an indication of how large fraction of the roadmap that really has to be explored, and the amount of work saved by our lazy approach, in this particular example. Note this is a conservative estimate since even with this long preprocessing, there is no guarantee that the remaining roadmap will contain a feasible path. Table 1.1(b) shows whether a collision-free path was found or not. We see in 1.1(a) that several enhancement steps are needed with Lazy PRM, thus indicating that node enhancement also is needed with basic PRM, and this will further increase running times and the number of collision checks.

7.2   INTERPRETATION OF RESULTS

We clearly see in Table 1.1(a) that collision checking represents the vast majority of the planning time (79%), but also that the graph building takes a lot of time (19%). Interestingly, the time spent on graph searching is negligible, about 2%. Although we carefully select the points to check for collision by frequently searching the roadmap for a shortest path, the total time spent on that is very short.

The initial roadmaps consist of $N_{init} = 10,000$ nodes in all experiments. We see in Table 1.1(b) that the number of collision checks required to explore one entire roadmap is of order 500,000. Table 1.1(a) shows, on the other hand, that Lazy PRM in average solves the different planning tasks in 92 to 693 collision checks. Thus, Lazy PRM only explores a small fraction, less than 0.1%, of the roadmap. This is the strength of the algorithm; to either find a collision-free path or to conclude that none exists in the roadmap by using a small number of collision checks.

We also see in Table 1.1(a) that a large percentage, 27%, of the total number of collision checks are actually performed on the collision-free solution paths, and are therefore inevitable. This large percentage can be explained by two reasons. First, the algorithm finds a sequence of collision-free nodes before edges are being checked. This prevents from planning local paths in dead ends and in regions from where no way out exists. Secondly, we check the edges along the path starting from both ends with increasing resolution and stop as soon as a collision occurs. The colliding edge is removed from the roadmap, and a new shortest path is found. Thus, we avoid using a local planner and instead keep a global view throughout the planning process. As a consequence, very

few edges – often only the edges along the final path – are checked with the finest resolution. This also makes the algorithm relatively insensitive to the resolution with which the paths are checked.

Since all of the nodes in the initial roadmap are uniformly distributed, the number of collision-free nodes found by basic PRM will give a good estimation of the relative size of $\mathcal{F}$. We see in Table 1.1(b) that for the tasks $A \to B$ and $E \to A$ approximately 40% of $\mathcal{C}$ is collision-free. For the other tasks approximately 30% of $\mathcal{C}$ is collision-free. As expected, the free part of $\mathcal{C}$ is reduced when the robot grasps a sheet of metal.

Furthermore, from the planner's point of view, the robot's tool includes both the gripper and possibly also a sheet of metal attached to it. If the tool is large and irregularly shaped, then its orientation becomes more important, whereas if the tool is small (e.g. the gripper only), the wrist motions of the robot, which basically determine the orientation, become less important. In this kind of environment, the planning problem is significantly easier if the tool is small. This explains why the tasks $A \to B$ and $E \to A$ are successfully planned without any node enhancement, and reveals the strength of our method in adapting to the difficulty of the problem.

8.   DISCUSSION

The aim of Lazy PRM is essentially to minimize the number of collision checks while searching for the shortest feasible path in a roadmap in the context of a PRM planner. This is done on the expense of frequent graph search. For a complex robot working in a complex workspace, like our six dof example, collision checking is an expensive operation, and careful selection of the points being checked for collision reduces the planning time considerably.

However, if the robot and the obstacles have a very simple geometry, then collision checking is very fast. Frequent graph searching may, instead of speeding up the planning, become a bottleneck. Trading some collision checking for less graph searching may increase performance of Lazy PRM. So, instead of re-planning the entire path every time a collision is found, we can try to remove several nodes from the roadmap in each iteration of the main loop, see Figure 1.2. A simple way would be to always check *all* nodes along a path before searching for a new path.

Another modification of Lazy PRM is necessary when the configuration space is very cluttered. This is, for instance, the case with the ten dof robot in [24], where more than 99% of the configuration space is infeasible. If we run our algorithm, we would need a large number of nodes in the initial roadmap, and then remove from the roadmap ap-

Table 1.1 Performance data for Lazy PRM based on 20 consecutive runs for each task. Table 1.1(b) shows data for basic PRM based on one run for each task. The initial number of nodes, N_{init}, is 10000 in all tests.

		Task				
		$A{\to}B$	$B{\to}C$	$D{\to}E$	$E{\to}A$	*Total*
Lazy PRM						
Number of collision checks						
-for nodes	ave	9	209	322	18	558(37%)
-for edges	ave	83	387	371	124	965(63%)
	min	74	169	151	81	
-total	ave	92	596	693	142	1523
	max	131	1114	1301	299	
-for returned path	ave	78	136	117	82	413(27%)
Number of enh. steps						
	min	0	0	0	0	
	ave	0	1.3	1.7	0	
	max	0	3	4	0	
Running time (sec.)						
-graph building	ave	6.6	7.9	8.5	6.6	29.6(19%)
-graph searching	ave	0	0.9	3.0	0	3.9(2%)
-coll. checking	ave	6.1	45.7	62.4	11.6	125.8(79%)
	min	11.2	19.1	21.4	13.0	
-total	ave	12.7	54.5	73.9	18.2	159.3
	max	16.2	102.7	133.5	31.2	

Table 1.1(a).

	$A{\to}B$	$B{\to}C$	$D{\to}E$	$E{\to}A$
PRM				
Number of collision checks				
-for nodes	10000	10000	10000	10000
of which in $\mathcal{F}$	4085	2980	3041	4121
-for edges	763063	419613	446782	787507
-total	773063	429613	456782	797507
Running time (sec.)				
-total	56625	32088	35115	56234
Found feasible path	yes	yes	no	yes

Table 1.1(b).

proximately 99% of the nodes being checked, which would take a lot of time. Fortunately, we can easily modify Lazy PRM to check all nodes *before* we insert them into the roadmap. This would certainly cause unnecessary nodes to be checked for collision, but, on the other hand, we would save many inserting and removing operations in the roadmap. After that, we still have the efficient way of exploring the edges along paths. In this way, the lazy approach can be employed with most of the existing sampling schemes and variations of PRM discussed in Section 4.2.

Our primary interest in this project has been path planning in industrial environments, and the experimental results show that Lazy PRM works well in practice. By using either or both of the two modifications of the algorithm suggested above, we can tune the amount of graph search according to the application and the time required to perform a collision check, so that Lazy PRM becomes efficient for an even wider range of problems.

As far as future work is concerned let us note the following. Lazy PRM has essentially one parameter that is critical for the performance – N_{init}, the initial number of nodes. As indicated in Theorem 1, N_{init} is strongly correlated to the probability of finding a feasible path without using the node enhancement step. The optimal choice depends on the dimension of $\mathcal{C}$, the workspace, the planning task, and the desired quality of the solution path. Our future work includes an investigation of the dependence between N_{init} and the planning time in different environments, as well as different distributions of the nodes.

Randomized techniques, like Lazy PRM, often give very fast planning. In Table 1.1(a), however, we can see that the maximum planning time is approximately twice as long as the average planning time. New improved enhancement techniques, in order to make the algorithms more robust in the sense that the worst case performance is improved, will also be a topic of our future research.

9.   SUMMARY

The advantage of considering the path planning problem in the configuration space is that even the most complicated robot is transformed into a single point. On the other hand, simple obstacles in the workspace generally become very complicated in the configuration space. Thus, we trade complex robots and simple obstacles for simple robots and complex obstacles. The new problem is essentially to explore an unknown space. But without a priori knowledge, or assumptions of the properties of the space, it is hard to construct powerful heuristic algorithms that

can solve the problem. This is where randomized techniques have shown to be very useful, particularly in high-dimensional configuration spaces. Randomized planners (like PRM and Lazy PRM) are generally easy to implement and very efficient in exploring unknown environments, which make them popular and applicable to a wide variety of problems.

In this paper we further develop randomized planning techniques in the direction of achieving general and practically useful single query planners. We address standard industrial applications characterized by complex geometry and high-dimensional, relatively uncluttered configuration spaces. Our algorithm – called Lazy PRM – is based upon a general scheme for lazy evaluation of the feasibility of the roadmap. The scheme is simple and general and can be applied to any graph that needs to be explored. In addition to Lazy PRM, most other existing variations of PRM, and other related algorithms, can benefit from this scheme and significantly increase performance.

Acknowledgments

References

[1] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1998.

[2] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 630–637. AK Peters, 1998.

[3] N.M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 113–120, 1996.

[4] J. Barraquand, L. E. Kavraki, J. C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *Int. J. of Robotics Research*, 16(6):759–775, 1997.

[5] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. of Rob. Research*, 10:628–

649, 1991.

[6] R. Bohlin. *Motion Planning for Industrial Robots*. Licentiate thesis, Chalmers University of Technology, 1999.

[7] R. Bohlin and L.E. Kavraki. A lazy probabilistic roadmap planner for single query path planning. Submitted to Int. J. on Robotics Research.

[8] R. Bohlin and L.E. Kavraki. Path planning using lazy PRM. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2000.

[9] V. Boor, M.H. Overmars, and F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1018–1023, 1999.

[10] L. Kavraki C. Holleman. A framework for using the workspace medial axis in PRM planners. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2000.

[11] S. Cameron. Enhancing GJK: Computing minimum distance and penetration distanses between convex polyhedra. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 3112–3117, 1997.

[12] J.F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[13] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1718–1723, 1990.

[14] A. Gray. *Tubes*. Addison-Wesley, Redwood City, CA, 1990.

[15] K. Gupta and A. P. del Pobil. *Practical Motion Planning in Robotics*. John Wiley, West Sussex, England, 1998.

[16] L. Han and N.M Amato. Kinematics-based probabilistic roadmap method for closed chain systems. In *Wokshop on the Algorithmic Foundations of Robotics*, 2000.

[17] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom - random reflections at C-space obstacles. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 3318–3323, 1994.

[18] D. Hsu, L.E. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P. Agarwal, L. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 141–154. A K Peters, 1998.

[19] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 2719–2726, 1997.

[20] Y.K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Comp. Surveys*, 24(3):219–291, 1992.

[21] P. Isto. A two-level search algorithm for motion planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 2025–2031, 1997.

[22] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, New Jersey, 1998.

[23] L.E. Kavraki, M.N. Kolountzakis, and J.C. Latombe. Analysis of probabilistic roadmaps for path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 3020–3025, 1996.

[24] L.E. Kavraki and J.C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 2138–2145, 1994.

[25] L.E. Kavraki, P. Švestka, J.C. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. & Aut.*, 12:566–580, 1996.

[26] J.J Kuffner and J.C. Latombe. Fast synthetic vision, memory, and learning models for virtual humans. In *IEEE Computer Animation*, pages 118 –127, 1999.

[27] J.C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[28] J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *Int. J. of Rob. Research*, 18(11):1119–1128, 1999.

[29] J.P. Laumond and T. Siméon. Notes on visibility roadmaps and path planning. In *Wokshop on the Algorithmic Foundations of Robotics*, 2000.

[30] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 473–479, 1999.

[31] S.M. LaValle, J.H. Yakey, and L.E. Kavraki. A proababilistic roadmap approach for systems with closed kinematic chains. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1671–1676, 1999.

[32] M.C. Lin and J.F. Canny. A fast algorithm for incremental distance computation. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1008–1014, 1991.

[33] G.F. Luger and W.A. Stubblefield. *Artificial intelligence and the design of expert systems*. Benjamin/Cummings, Redwood City, CA, 1989.

[34] A.H. Mishkin, J.C. Morrison, T.T. Nguyen, B.K. Cooper H.W. Stone, and B.H. Wilcox. Experiences with operations and

autonomy of the mars pathfinder microrover. In *IEEE Aerospace Conference*, pages 337 –351, 1998.

[35] C.L. Nielsen and L.E. Kavraki. A two level fuzzy PRM for manipulation planning. Technical Report TR2000-365, Rice University, 2000.

[36] M. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Utrecht University, the Netherlands, 1992.

[37] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In K.Y. Goldberg, D. Halperin, J.C. Latombe, and R.H. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 19–37. A K Peters, 1995.

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

[39] F. Thomas and C. Torras. Interference detection between non-convex polyhedra revisited with a practical aim. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1994.

[40] R.Z. Tombropoulos, J.R. Adler, and J.C. Latombe. CARABEAMER: A treatment planner for a robotic radiosurgical system with general kinematics. *Medical Image Analysis*, 3(3):237–264, 1999.

[41] S.A. Wilmarth, N.M. Amato, and P.F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1024–1031, 1999.