Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces

Lydia Kavraki[1] Petr Švestka[2]
Jean-Claude Latombe[1] Mark Overmars[2]

Abstract

A new motion planning method for robots in static workspaces is presented. This method proceeds according to two phases: a learning phase and a query phase. In the learning phase, a probabilistic roadmap is constructed and stored as a graph whose nodes correspond to collision-free configurations and edges to feasible paths between these configurations. These paths are computed using a simple and fast local planner. In the query phase, any given start and goal configurations of the robot are connected to two nodes of the roadmap; the roadmap is then searched for a path joining these two nodes. The method is general and easy to implement. It can be applied to virtually any type of holonomic robot. It requires selecting certain parameters (e.g., the duration of the learning phase) whose values depend on the considered scenes, that is the robots and their workspaces. But these values turn out to be relatively easy to choose. Increased efficiency can also be achieved by tailoring some components of the method (e.g., the local planner) to the considered robots. In this paper the method is applied to planar articulated robots with many degrees of freedom. Experimental results show that path planning can be done in a fraction of a second on a contemporary workstation ($\approx$ 150 MIPS), after learning for relatively short periods of time (a few dozen seconds).

Acknowledgments: L. Kavraki and J.C. Latombe were partially supported by ARPA grant N00014-92-J-1809 and ONR grant N00014-94-1-0721. L. Kavraki also acknowledges the support of the Rockwell Foundation. P. Švestka and M. Overmars were partially supported by ESPRIT III BRA Project 6546 (PROMotion) and by the Dutch Organization for Scientific Research (NWO).

[1]Robotics Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305, USA.

[2]Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands.

1 Introduction

We present a new planning method which computes collision-free paths for robots of virtually any type moving among stationary obstacles (static workspaces). However, our method is particularly interesting for robots with many degrees of freedom (dof), say five or more. Indeed, an increasing number of practical problems involve such robots, while very few effective motion planning methods, if any, are available to solve them. The method proceeds according to two phases: a *learning phase* and a *query phase*.

In the learning phase a probabilistic *roadmap* is constructed by repeatedly generating random free configurations of the robot and trying to connect 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 R. The configurations are the nodes of R and the paths computed by the local planner are the edges of R. The learning phase is concluded by some postprocessing of R to improve its connectivity.

Following the learning phase, multiple queries can be answered. A *query* asks for a path between two given free configurations of the robot. To process a query the method first attempts to connect the given start and goal configurations to two nodes of the roadmap, with paths that are feasible for the robot. Next, a graph search is done to find a sequence of edges connecting these nodes in the roadmap. Concatenation of the successive path segments transforms this sequence into a feasible path for the robot.

Notice that the learning and the query phases do not have to be executed sequentially. Instead, they can be interwoven to adapt the size of the roadmap to difficulties encountered during the query phase, thus increasing the learning flavor of our method. For instance, a small roadmap could be first constructed; this 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 learning phase.

To run our planning method the values of several parameters must first be selected, e.g., the time to be spent in the learning phase. While these values depend on the considered *scene*, i.e., the robot and the workspace, it has been our experience that good results are obtained with values spanning rather large intervals. Thus, it is not difficult to choose one set of satisfactory values for a given scene or family of scenes, through some preliminary experiments. Moreover, increased efficiency can be achieved by tailoring several components of our planning method, in particular the local planner, to the considered robots. Overall, we found the method quite easy to implement and run. Many details can be engineered in one way or another to fit better the characteristics of an application domain.

We have demonstrated the power of our method by applying it to a number of difficult motion planning problems involving a variety of robots. In this paper we report in detail on experiments with planar articulated robots (or linkages) with many dofs moving in constrained workspaces. However, the method is directly applicable to other kinds of holonomic robots, such as spatial articulated robots in 3D workspaces [KL94b]. Additionally, a version of the method described here has been successfully applied to nonholonomic car-like robots [ŠO94]. In all cases, experimental results show that the learning times required for the construction of adequate roadmaps, i.e., roadmaps that capture well the connectivity of the free C-space, are low. They range from a few seconds[1] for relatively easy problems to a few minutes for the most difficult problems we have dealt with. Once a good roadmap has been constructed, path planning queries are processed in a fraction of a second.

The very small query times make our planning method particularly suitable for many-dof robots performing several point-to-point motions in known static workspaces. Examples of tasks meeting these conditions include maintenance of cooling pipes in a nuclear plant, point-to-point welding in car assembly, and cleaning of airplane fuselages. In such tasks, many dofs are needed to achieve successive desired configurations of the end-effector while avoiding collisions of the rest of the arm with the complicated workspace. Explicit programming of such robots is tedious and time consuming. An efficient and reliable planner would considerably reduce the programming burden.

This paper is organized as follows: Section 2 gives an overview of some previous research and relates our work to this research. Section 3 describes our motion planning method in general terms, i.e., without focusing on any specific type of holonomic robot. Both the learning phase and the query phase are discussed here in detail. Next, in Sections 4, 5, and 6 we apply our method to planar articulated robots. In Section 4 we describe specific techniques that can be substituted for more general ones in the planner to handle these robots more efficiently (especially when these have many dofs). In Sections 5 and 6 we describe a number of experiments and their results; we also analyze how variations of some parameter values affect planning results. Section 5 presents results obtained with a customized implementation of the method embedding the specific techniques of Section 4. Section 6 discusses other experimental results obtained with a general implementation of the method. Section 7 concludes the paper.

[1]All running times reported in this paper have been obtained on a DEC Alpha workstation, except those given in Section 6 which were obtained with a Silicon Graphics Indigo workstation.

2 Relation to previous work

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 heuristic, but practical path planners. For few-dof robots, many such planners have been designed and some are extremely fast (e.g., [BLL92, LRDG90]). Considerable attention is now directed toward the creation of efficient heuristic planners for many-dof robots. Indeed, while such robots are becoming increasingly useful in industrial applications, complete methods in that case have overwhelming complexity. New emerging applications also motivate that trend, e.g., computer graphic animation, where motion planning can drastically reduce the amount of data input by human animators, and molecular biology, where motion planning can be used to compute motions of molecules (modeled as spatial linkages with many dofs) docking against other molecules.

The complexity of complete path planning methods in high-dimensional configuration spaces has led researchers to seek heuristic methods that embed weaker notions of completeness (e.g., probabilistic completeness) and/or can be partially adapted to specific problem domains in order to boost performance in those domains.

In recent years, some of the most impressive results were obtained using potential field planning methods. Such methods are indeed attractive, since the main heuristic function they use to guide the search for a path, the potential field, can easily be adapted to the specific problem to be solved, in particular the scene and the goal configuration. Two main lines of research are particularly noteworthy:

- A method using a "dynamic" potential field is proposed in [FT87] for planning the paths of robots with many dofs. The potential function depends not only on the distance between the robot and the obstacles, but also on the rate of variation of this distance along the current direction of motion of the robot. The method can be very fast on rather simple examples, but it may get stuck at local minima of the potential function on more difficult ones. It was used to compute paths of an 8-dof manipulator among vertical pipes in a nuclear plant, with interactive human assistance to escape local minima. In [FT90] the same authors present a learning scheme to avoid falling into local minima. During the learning phase, probabilities of moving between neighboring configurations without falling into a local minimum are accumulated in an r^n array, where n is the number of dofs and r is the number of intervals discretizing the range of each dof. During the planning phase, these probabilities are used as another heuristic function (in addition to the potential function) to guide the robot away from the local minima. This learning scheme was applied with some success to robots with up to 6 dofs. However, the size of the r^n array becomes impractical when n grows larger.

- Techniques for both computing potential functions and escaping local minima in high-dimensional C-spaces are presented in [BL91, BLL92]. The Randomized Path Planner (RPP) described in [BL91] escapes local minima by executing random walks. It has been successfully experimented on difficult problems involving robots with 3 to 31 dofs. It has also been used in practice with good results to plan motions for performing riveting operations on plane fuselages [GMKL92]. Recently, RPP has been embedded in a larger "manipulation planner" to automatically animate graphic scenes involving human figures modeled with 62 dofs [KKKL94]. However, several examples have also been identified where RPP behaves poorly [CG93, ZG93]. 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. In fact, once one knows how RPP computes the potential field, it is not too difficult to create such examples. One way to prevent this from happening is to let RPP randomly use several potential functions, but this solution is rather time consuming. In [BF94] a very promising method based variational dynamic programming is presented and that method can tackle problems of similar complexity to the problems solved by RPP.

Other interesting lines of work include the following: In [GG92, 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 six-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].

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, our method applies a roadmap approach [Lat91], that is, it constructs a network of paths in free C-space. Previous roadmap methods include the visibility graph [LPW79], Voronoi diagram [OY82], and silhouette [Can88] methods. All these three methods compute in a single shot a roadmap that completely represents the connectivity of the free C-space. But the visibility graph and Voronoi diagram methods are limited to low-dimensional C-spaces. In theory the silhouette method applies to C-spaces of any dimension, but its complexity makes it little practical. In contrast, our method builds a roadmap *incrementally* using probabilistic techniques. These techniques apply to C-spaces of any dimension and produce a roadmap in any amount of time allocated to them. Of course, if this time is too short, the computed roadmap may not represent the connectivity of free C-space well. Actually, in our planner, the roadmap is never guaranteed to fully represent free C-space connectivity, though if we let our techniques run long enough it eventually will (but we don't know how long is enough). However, while building the roadmap, our method heuristically identifies "difficult" regions in free C-space and generates additional configurations in those regions to increase network connectivity. Therefore, the final distribution of configurations in the roadmap is not uniform across free C-space; it is denser in regions considered difficult by the heuristic function. This feature helps to construct roadmaps of reasonable size that represent free C-space connectivity well. In particular, it allows our implemented planner to efficiently solve tricky problems requiring choices among several narrow passages, i.e., the kind of problems that RPP tackles poorly.

Note also that, like most practical methods for many-dof robots (one exception is the method in [FT89]), RPP is a one-shot method, i.e., it does not precompute any knowledge of the free C-space that is transferred from one run to another. Consequently, on problems that both RPP and our method solve well, the latter is usually much faster, once it has constructed a good roadmap. But, if the learning time is included in the duration of the path planning process (which should be the case whenever planning is done only once in a given workspace), there are many problems for which RPP is faster.

The authors of this paper are from two different teams and the work presented here builds upon previous work they did separately. A single-shot random planner was described in [Ove92] and was subsequently expanded into a learning approach in [OŠ94]. In these papers the emphasis was on robots with a rather low number of dofs. Similar techniques have been applied both to car-like robots that can move forward and backward (symmetrical nonholonomic robots) and car-like robots that can only move forward [Šve93, ŠO94]. Independently, a preprocessing scheme similar to the learning phase was introduced in [KL93] for planning the paths of many-dof robots. This scheme also builds a probabilistic roadmap in free C-space, but focuses on the case of many-dof robots. The need to expand the roadmap in "difficult" regions of C-space was noted there and addressed with simple techniques. Better expansion techniques were introduced in [KL94a, KL94b]. The present paper combines the ideas of these previous papers and extends them into a more powerful and faster planner. Since it only presents a limited subset of the experimental results we have obtained with our method, the interested reader is encouraged to look into our previous papers for additional results, in particular results involving other types of robots. Though computation times reported in these papers were obtained with previous versions of our method, their orders of magnitude remain meaningful.

Finally, it should be noted that another planner which bares similarities with our approach, but was developed independently of our two teams, is proposed in [HST94].

3 The general method

We now describe our path planning method in general terms for a holonomic robot without focusing on any specific type of robot. During the learning phase a data structure called the roadmap is constructed in a probabilistic way for a given scene, i.e., a given robot and a given workspace. In the query phase, the roadmap is used to solve individual path planning problems in this scene. Each problem is specified by a start configuration and a goal configuration of the robot.

The roadmap is constructed as an undirected graph $R = (N, E)$. The nodes in N are randomly generated free configurations of the robot and the edges in E correspond to (simple) paths; an edge (a, b) corresponds to a feasible path connecting the configurations a and b. These paths, which we refer to as local paths, are computed by an extremely fast, though not very powerful planner, called the local planner. The local paths are not explicitly stored in the roadmap, since recomputing them is very cheap. This saves considerable space, but requires the local planner to succeed and fail deterministically. We assume here that the learning phase is entirely performed before any path planning query is processed. However, as we already noted, the learning and query phases could also be interwoven.

In the query phase, given a start configuration s and a goal configuration g, the method first tries to connect s and g to some two nodes $\tilde{s}$ and $\tilde{g}$ in N. If successful, it then searches R for a sequence of edges in E connecting $\tilde{s}$ to $\tilde{g}$. Finally, it transforms this sequence into a feasible path for the robot by recomputing the corresponding local paths and concatenating them.

In the following, we let $\mathcal{C}$ denote the robot's C-space and $\mathcal{C}_f$ its free subset (also called the free C-space).

3.1 The learning phase

The learning phase consists of two successive steps, which we refer to as the construction and the expansion step. The objective of the former is to obtain a reasonably connected graph, with enough vertices to provide a rather uniform covering of free C-space and make sure that most "difficult" regions in this space contain at least a few nodes. The second 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 C-space and expand the graph around these nodes by generating additional nodes in their neighborhoods. Hence, the covering of free C-space by the final roadmap is not uniform, but depends on the local intricacy of that space.

3.1.1 The construction step

Initially the graph $R = (N, E)$ is empty, i.e., $N = E = \emptyset$. Then, repeatedly, a random free configuration is generated and added to N. For every such new node c, we select a number of nodes from the current N and we try to connect c to each of them using the local planner. Whenever this planner succeeds to compute a feasible path between c and a selected node n, the edge (c, n) is added to E. The actual local path is not memorized.

The selection of the nodes to which we try to connect c is done as follows: First, a set N_c of candidate neighbors is chosen from N. This set is made of nodes within a certain distance of c, for some metric D. Then we pick nodes from N_c in order of increasing distance from c. We try to connect c to each of the selected nodes if it is not already graph-connected to c. Hence, no cycles can be created and the resulting graph is a forest, i.e., a collection of trees. Since a query would never succeed *thanks to* an edge that is part of a cycle, it is indeed sensible not to consume time and space computing and storing such an edge. However, in some cases, the absence of cycles may lead the query phase to construct unnecessary long paths. This drawback can easily be eliminated by applying smoothing techniques to either the roadmap during the learning phase, or the particular paths constructed by the query phase, or both. Even if the roadmap contained cycles, such smoothing operations would eventually produce better paths.

Whenever the local planner succeeds to find a path between two nodes, the connected components of R are dynamically updated. Therefore, no graph search is required for deciding whether a node picked from N_c is already connected to c, or not.

To make our presentation more precise, let:

- Δ be a symmetrical function $\mathcal{C}_f \times \mathcal{C}_f \to \{0, 1\}$, which returns whether the local planner can compute a feasible path between the two free configurations given as arguments;
- D be a function $\mathcal{C} \times \mathcal{C} \to R^+ \cup \{0\}$, called the *distance function*, defining

a pseudo-metric in $\mathcal{C}$. (We only require that D be symmetrical and non-degenerate.)

The construction step algorithm can now be outlined as follows:

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

This outline leaves a number of components unspecified. Indeed, we still must define how random configurations are created in (4), propose a local planner for (8), clarify the notion of a candidate neighbor in (5), and choose the distance function D used in (7).

Creation of random configurations. The nodes of R should constitute a rather uniform random sampling of $\mathcal{C}_f$. Every such configuration is obtained by drawing each of its coordinates from the interval of 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.

Collision checking requires testing if any part of the robot intersects an obstacle and if two distinct bodies of the robot intersect each other. It can be done using a variety of existing general techniques. In the general implementation considered in Section 6 the test is performed analytically using optimized routines from the PLA-GEO library [Gie93]. Alternatively, we could use an iterative collision checker, like the one described in [Qui93], which automatically generates successive approximations of the objects involved in the collision test. In 2D workspaces, we may use a faster, but more specific collision checker (see Section 4).

The local planner. The local planner should be both deterministic and very fast. These requirements are not strict, however.

If a non-deterministic planner was used instead, local paths would simply have to be stored in the roadmap. The roadmap would require more space, but this would not be a major problem.

Concerning how fast the local planner should be, there is clearly a tradeoff between the time spent in each individual call of this planner and the number of calls. If a powerful local planner was used, it would often succeed in finding a path when one exists. Hence, relatively few nodes would be required to build a roadmap capturing the connectivity of the free C-space sufficiently well to reliably answer path planning queries. Such a local planner would probably be rather slow, but this could be somewhat compensated by the small number of calls needed. On the other hand, a very fast planner is likely to be less successful. It will require more configurations to be included in the roadmap; so, it will be called more often, but each call will be cheaper.

The choice of the local planner also affects the query phase. The purpose of having a learning phase is to make it possible to answer path planning queries quasi-instantaneously. It is thus important to be able to connect any given start and goal configurations to the roadmap, or to detect that no such connection is possible, very quickly. This requires that the roadmap be dense enough, so that it always contains a few nodes (at least one) to which it is easy to connect each of the start and goal configurations. It thus seems preferable to use a very fast local planner, even if it is not too powerful, and build large roadmaps with configurations widely distributed over free C-space. We actually tried several local planners, some very fast, some slower but more powerful, and our experimental observations clearly confirmed this conclusion (e.g., see [Mas92, Šve93]).

Choosing a very fast local planner for the learning phase has two other advantages. First, the same local planner can then be used during the query phase to connect the start and goal configurations to the roadmap. Second, local paths do not have to be memorized in the roadmap.

A quite general such local planner, which is applicable to all holonomic robots, connects any two given configurations by a straight line segment in configuration space and checks this line segment for collision and joint limits (if any). Verifying that a straight line segment remains within joint limits is straightforward. On the other hand, collision checking can be done as follows [BL91]: First, discretize the line segment (more generally, any path generated by the local planner) into a number of configurations $c_1, \ldots, c_m$, such that for each pair of consecutive configurations (c_i, c_{i+1}) no point on the robot, when positioned at configuration c_i, lies further than some `eps` away from its position when the robot is at configuration c_{i+1} (`eps` is an input positive constant).[2] Then, for each configuration c_i, test whether the robot, when positioned at c_i and "grown" by `eps`, is collision-free, using the collision checker discussed above. If none of the m configurations yield collision, conclude that the path is collision-free. Since `eps` is constant, the computation of the robot bodies grown by `eps` is done only once. In the following we will refer to this local planner as the *general local planner*.

The node neighbors. Another important choice to be made is that of the candidate neighbors of a node c. The definition of the set N_c considerably affects the performance of the construction step because, together, the executions of the local planner form the single most time-consuming operation at this step.

We must thus prevent executions of the local planner that do not lead to effectively extending the knowledge stored in the roadmap. First, as mentioned before, we do not try to connect configurations that are already in the same connected component of the roadmap. Second, we try to 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:

$$N_c \subseteq \{\tilde{c} \in N | D(c, \tilde{c}) \leq \texttt{maxdist}\}.$$

This still leaves several possibilities for the actual definition of N_c. We have done experiments with different definitions and the following one gives good results over a wide range of problems. We consider as candidate neighbors of c all nodes in N within distance `maxdist` of c. That is, according to the algorithm outline given above, we try to connect c to all nodes in the neighborhood of c defined by `maxdist`, in order of increasing distance from c; but we skip those nodes which are in the same connected component c at the time the connection is to be tried. By considering elements of N_c in this order we expect to maximize the chances of quickly connecting c 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 useful to bound the size of the set N_c by some constant `maxneighbors` (typically on the order of 30). 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 R. Thus, the construction step takes linear time in the size of the graph it constructs.

[2] Throughout this paper symbols in teletyped characters are used to denote parameters of the planning method.

The distance function. The function D is used to both construct and sort the set N_c of candidate neighbors of each new node c. It should be defined so that, for any pair (c, n) of configurations, $D(c, n)$ reflects the chance that the local planner will *fail* to compute a feasible path between these configurations. One possibility is thus to define $D(c, n)$ 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 c and n in the absence of obstacles. Thus, each local planner would automatically induce its own specific distance function. In general, though, exact computation of swept areas/volumes tends to be rather time-consuming. Instead, rough but inexpensive-to-evaluate approximations of the swept-region measure or functions that vary approximately like this measure give better practical results. For example, when the general local planner described above is used to connect c and n, $D(c, n)$ may be defined as the longest Euclidean distance that any point on the robot travels in workspace, when the robot moves along the line segment joining c and n in configuration space, i.e.:

$$D(c, n) = \max_{x \in \text{robot}} \|x(n) - x(c)\|, \tag{1}$$

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

3.1.2 The expansion step

If the number of nodes generated during the construction step is large enough, the set N gives a fairly uniform covering of the free C-space. In easy scenes R is then well connected. But in more constrained ones where free C-space is actually connected, R often consists of a few large components and several small ones. It therefore does not effectively capture the connectivity of $\mathcal{C}_f$. More generally, the number of large components in R usually exceeds the number of connected components in $\mathcal{C}_f$; and R also contains an even larger number of very small components. We have frequently observed this situation in our experiments.

The expansion step is intended to improve the connectivity of the graph R generated by the construction step. Typically, if the graph is disconnected in a place where $\mathcal{C}_f$ is not, this place corresponds to some narrow, hence difficult region of the free C-space. The idea underlying the expansion step is to select a number of nodes from N which are likely to lie in such regions and to "expand" them. By expanding a configuration c, we mean selecting a new free configuration in the neighborhood of c, adding this configuration to N, and trying to connect it to other nodes of N, in the same way as in the construction step. So, the expansion step increases the density of roadmap configurations in regions of $\mathcal{C}_f$ that are believed to be difficult. Since the "gaps" between components of the graph R are typically located in these regions, the connectivity of R is likely to increase.

We propose the following probabilistic scheme for the expansion step. With each node c in N we associate a positive weight $w(c)$ that is a heuristic measure of the "difficulty" of the region around c. Thus, $w(c)$ is large whenever c is considered to be in a difficult region. We normalize w so that all weights together (for all nodes in N) add up to one. Then, repeatedly, we select a node c from N with probability:

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

and we expand this node.

It now remains to define the heuristic weight $w(c)$. One possibility is to count the number of nodes of N lying within some predefined distance of c. If this number is low, the obstacle region probably occupies a large subset of c's neighborhood. This suggests that $w(c)$ could be defined inversely proportional to the number of nodes within some distance of c. Another possibility is to look at the distance d_c from c to the nearest connected component not containing c. If this distance is small, then c lies in a region where two components failed to connect, which indicates that this region might be a difficult one (it may also be actually obstructed). This idea leads to defining $w(c)$ inversely proportional to d_c. Alternatively, rather than using the structure of R to identify difficult regions, we could define $w(c)$ according to the behavior of the local planner. For example, if the local planner often failed to connect c to other nodes, this is also an indication that c lies in a difficult region. Which particular heuristic function should be used depends to some extent on the input scene. Nevertheless, the following function, which is based on the latter idea, has produced good results whenever we tried it:

- During the construction step, for each new node c, compute the failure ratio $r_f(c)$ defined by:

  $$r_f(c) = \frac{f(c)}{n(c)+1},$$

  where $n(c)$ is total number of times the local planner tried to connect c to another node and $f(c)$ is the number of times it failed. (Note: Whenever the local planner fails to connect two nodes c and n, this failure is counted in *both* the failure ratios of c and n. In this way, the configurations that are included in N at the very beginning of the construction step get meaningful failure ratios.)

- At the beginning of the expansion step, for every node c in N compute $w(c)$ proportional to the failure ratio, but scaled appropriately so that all weights add up to one, i.e.:

$$w(c) = \frac{r_f(c)}{\sum_{a \in N} r_f(a)}.$$

Once we have decided which nodes to expand, we have to choose how to perform this expansion. We have done experiments with different techniques and we have finally selected a technique which makes use of what we call random-bounce walks (or rbw). For holonomic robots, an rbw consists of repeatedly picking at random a direction of motion in C-space and moving in this direction until an obstacle is hit. When a collision occurs, a new random direction is chosen. And so on. To expand a node c, we compute one rbw starting from c. We limit the computation time, i.e., the duration of the rbw, to a short amount (say, 0.01 seconds). The final configuration n reached by the rbw and the edge (c, n) are included into R. Moreover, the path computed between c and n is explicitly stored, since it was generated by a non-deterministic technique. We also record the fact that n belongs to the same connected component as c. Then we try to connect n 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 reducing the number of components.

The weights $w(c)$ are computed only once at the beginning of the expansion step and are not modified when new nodes are added to R. Hence, the nodes to expand are all selected from the set of nodes generated during the construction step. Alternatively, we could update the weights whenever the expansion step inserts a new node into N. We believe that the potential gain of recomputing weights is largely offset by the time it requires.

Once the expansion step is over, the remaining small components of R, if any, are discarded. Here, a component is considered small if its number of nodes is less than some `mincomponent` percent (typically 0.01%) of the total number of nodes in N. The graph R after discarding the small components represents the roadmap that will be used during the query phase. It may contain one or several components.

Let $\mathtt{T_L}$ be the time allocated to the learning phase, i.e. the computation of the roadmap. Clearly, the range of adequate values for $\mathtt{T_L}$ depends on the considered scene, so that an adequate value should be determined experimentally for each new scene. Another important parameter is how $\mathtt{T_L}$ is divided between the construction step (time $\mathtt{T_C}$) and the expansion step (time $\mathtt{T_E}$). Our experience is that a 2:1 ratio, i.e, $\mathtt{T_C} = 2\mathtt{T_L}/3$ and $\mathtt{T_E} = \mathtt{T_L}/3$, gives good results over a large range of problems.

3.2 The query phase

During the query phase, paths are to be found between arbitrary input start and goal configurations, using the roadmap constructed in the learning phase. Assume for the moment that the free C-space is connected and that the roadmap consists of a single connected component R. A query now consists of the following: Given a start configuration s and goal configuration g, we try to connect s and g to some two nodes of R, respectively $\tilde{s}$ and $\tilde{g}$, with feasible paths P_s and P_g. If this fails, the query fails. Otherwise, we compute a path P in R connecting $\tilde{s}$ to $\tilde{g}$. A feasible path from s to g is eventually constructed by concatenating P_s, the local paths recomputed by the local planner when applied to pairs of consecutive nodes in P, and P_g reversed. If one wishes, this path may be improved by running a smoothing algorithm on it.

The main question is how to compute the paths P_s and P_g. The queries should preferably terminate quasi-instantaneously, so no expensive algorithm is desired here. Our strategy for connecting s to R is to consider the nodes in R in order of increasing distance from s (according to D) and try to connect s to each of them with the local planner, until one connection succeeds. We ignore nodes located further than `maxdist` away from s, because we consider that the chance of success of the local planner is too low. If all connection attempts fail, we perform one or more random-bounce walks, as described in Subsection 3.1.2. But, instead of adding the node at the end of each such rbw to the roadmap, we now try to connect it to R with the local planner. As soon as s is successfully connected to R, we apply the same procedure to connect g to R.

In general, however, the roadmap may consist of several connected components R_i, $i = 1, 2, \dots, p$. This is usually the case when the free C-space is itself not connected. It may also happen when free C-space is connected, for instance if the roadmap is not dense enough. If the roadmap contains several components, we first try to connect both the start and goal configurations s and g to two nodes in the *same* component. To do this, we consider the components of the roadmap in order of increasing distance from $\{s, g\}$; for each component we proceed as we did above with the single component R. We define the distance between $\{s, g\}$ and a component R_i as follows: Let the distance $D(c, R_i)$ between a configuration c and R_i be the minimum of $D(c, n)$ for all $n \in R_i$. The distance between $\{s, g\}$ and R_i is the maximum of $D(s, R_i)$ and $D(g, R_i)$. If the connection of s and g to some component R_i succeeds, a path is constructed as in the single-component case. The method returns failure whenever it fails to connect both s and g to the same roadmap component. Since in most examples the roadmap consists of rather few components, failure is rapidly detected.

Finally, we should note that certain kinds of local planners render unnecessary the recomputation of collisions along the network edges when the corresponding paths are reconstructed. This makes the planning stage even faster. For example, the general local planner of Subsection 3.1.1 aborts when a collision is detected. During planning time, intermediate configurations on a path induced by this planner have to be recomputed, since they have not been stored, but we do not need to check each of them for collision. The situation is different if the local planner does not abort when a collision is detected but performs a certain action. Then, in the planning stage collision must be checked along the recomputed path so that the same action can be repeated just after the collision is detected.

If path planning queries fail frequently, this is an indication that the roadmap may not adequately capture the connectivity of the free C-space. Hence, more time should be spent on the learning phase, i.e., $\mathtt{T_L}$ should be increased. However, it is not necessary to construct a new roadmap from scratch. Since the learning 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 learning and the query phases.

4 Application to planar articulated robots

This section and the next two describe the application of our planning method to planar articulated robots with fixed or free bases. In this section we present techniques specific to these robots that can be substituted for more general techniques in the planning method in order to increase its efficiency. The purpose of this presentation is to illustrate the easiness with which the general method for holonomic robots can be engineered to better suit the needs of a particular application. Many other specific tunings, not discussed here, are possible. In Section 5 we will discuss experiments with an implementation of the method that embeds the specific techniques described below, while in Section 6 we will present experimental results with a general implementation of the method to demonstrate that the method remains quite powerful, even without specific components. In the rest of the paper we will refer to these two implementations as the *customized implementation* and the *general implementation*, respectively.

To make the following presentation shorter, we consider planar articulated robots with revolute joints only, in arbitrary number. Figure 1 illustrates such a robot in which the links are line segments. The links, which may actually be any polygons, are denoted by L_1 through L_q (in the figure, $q = 5$). Points J_2 through J_q designate revolute joints. Point J_1 denotes the base of the robot; it may, or may not, be fixed relative to the workspace. If it is fixed, then J_1 is also a revolute joint. If it is not,

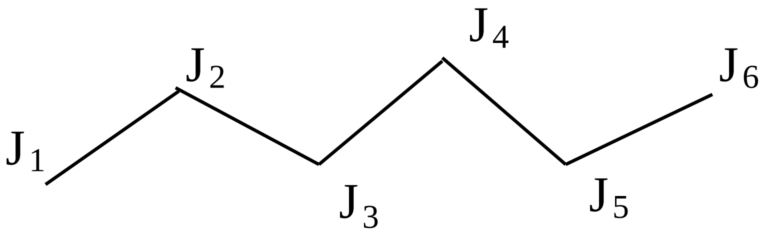

Figure 1: A planar articulated robot.

then J_1 can translate freely in the plane and the robot is said to have a free base. The point J_{q+1} (J_6 in the figure) is called the endpoint of the robot; actually, it is any point on the last link, preferably the one located the furthest away from J_q. Similarly, if the robot's base is free, J_1 can be any point on L_1, preferably the one located the furthest away from J_2. Each revolute joint J_i ($i = 1$ or 2 to q) has defined certain internal joint limits, denoted by low_i and up_i, with $low_i < up_i$, which constrain the range of the possible orientations that L_i can take relative to L_{i-1}. 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 by low_x and up_x, and low_y and up_y, respectively.

We represent the C-space of such a q-link planar articulated robot by:

$$[low_1, up_1] \times [low_2, up_2] \times \ldots \times [low_k, up_k],$$

if its base is fixed, and by:

$$[low_x, up_x] \times [low_y, up_y] \times [0, 2\pi] \times [low_2, up_2] \times [low_3, up_3] \times \ldots \times [low_k, up_k],$$

if its base is free. We call a self-collision configuration any configuration where two non-adjacent links of the robot intersect each other. We may, or may not allow such configurations. If we do not allow them, as is the case in all the examples considered in this paper, the free C-space is not only constrained by the obstacles, but also by the set of self-collision configurations. We assume that the joint limits prevent self-collisions between any two adjacent links.

We now discuss specific techniques for local path planning, distance computation, and collision checking that apply well to the family of robots defined above. The same techniques can also be applied, possibly with minor adaptations, to other types of articulated robots, e.g., robots with prismatic joints and/or with multiple kinematic chains (see [KL94a]) and articulated robots in 3D workspace (see [KL94b]).

Local path planning. Let a and b be any two given configurations that we wish to connect with the local planner. The local planner we use constructs a path as follows: It translates at constant relative velocity all the joints with an even index, i.e., all J_{2*i}'s, along the straight lines in the workspace that connect their positions at configuration a to their positions at configuration b. During this motion the planner adjusts the position of every other joint J_{2*i+1} using the straightforward inverse kinematic equations of this point relative to J_{2*i} and $J_{2*(i+1)}$. Thus, the J_{2*i+1}'s "follow" the motion led by the J_{2*i}'s. If q is odd, the position of J_q is not determined by the above rule; it is then computed by rotating joint J_q at constant revolute velocity relative to the linear velocity of point J_q. Recall from Subsection 3.1.1 that a local path is discretized into a sequence of configurations for collision checking. When our specific technique is used, we must also verify that the coordinates of each such configuration are within joint limits. Thus, the motion is aborted if either a collision occurs, or a joint moves beyond one of its limits, or a point J_{2*i+1} cannot follow the motion led by the J_{1*i}'s. We have observed that in cases when the above motion does not manage to connect configurations a and b, it nevertheless brings the robot to a configuration b' very close to b. It then pays off to try to connect b' and b with a straight line in C-space and only after this fails to declare failure of the local planner to connect a and b. In the following we will refer to the above planner as the *specific local planner*.

The workspace region swept out by the robot along a local path computed by the specific local planner between two configurations a and b is typically smaller than for the path joining a and b by a straight line segment in configuration space, which is computed by the general local planner described in Subsection 3.1.1. Hence, the local paths generated by the specific planner are more likely to be collision-free than those generated by the general planner. Also, collision checking is less expensive since, for a given `eps`, the discretization of the local path yields less configurations. On the other hand, the specific planner, though still very fast, is not as fast as the general planner. Indeed, checking that the dofs remain within joint limits along the local path requires inverse kinematic computation to determine configuration coordinates along the path. Furthermore, this check is not as rigorous, since it is performed only at a finite number of configurations. Nevertheless, our experience has been that the overall planning method performs significantly better on examples involving many-dof planar articulated robots, when the specific local planner is used instead of the general one.

Distance computation. In association with the above local planning technique we propose the following distance function D in configuration space: Let $J_i(a)$, $i = 1, \ldots, q+1$ denote the position of the point J_i in the workspace, when the robot is at configuration a. We define D by:

$$(a,b) \in \mathcal{C} \times \mathcal{C} \quad \mapsto \quad D(a,b) = \left(\sum_{i=1}^{q+1} \|J_i(a) - J_i(b)\|^2 \right)^{1/2},$$

where $\|J_i(x) - J_i(y)\|$ is the Euclidean distance between $J_i(a)$ and $J_i(b)$. When the robot has a fixed base, the first term of the above sum is zero. This function is a better approximation of the area swept by the robot along the local paths computed by the specific local planner than the general distance function defined by Equ. (1).

Collision checking. The 2D workspace allows for a very fast collision checking technique. In this technique each link of the robot is regarded as a distinct robot with two dofs of translation and one dof of rotation. A bitmap representing the 3D configuration space of this robot is precomputed, with the "0"'s describing the free subset of this space and the "1"'s describing the subset where the link collides with an obstacle. When a configuration is checked for collision, the 3D configuration of each link is computed and tested against its C-space bitmap, which is a constant-time operation. The configuration of a link is particularly fast to compute when the specific local planner is used, since this planner directly provides the coordinates of two points in the link. Note that we need not always create one bitmap for each link of the robot. For example, when all the links are line segments (as in Figure 1), a single bitmap can be computed, for the shortest link, by modeling the longer links as two (or more) short line segments. However, collision checking for a long link then requires multiple access to the bitmap.

The 3D bitmap for one link can be computed as a collection of 2D bitmaps, each corresponding to a fixed orientation of the link. If the link and the obstacles are modeled as collections of possibly overlapping convex polygons, the construction of a 2D bitmap can be done as follows [LRDG90]: First use the algorithm in [LP83] to produce the vertices of the obstacles in the link's C-space. (This algorithm takes linear time in the number of vertices of the objects.) Then draw and fill the obstacles into the 2D bitmap. (On many workstations, this second operation can be done very quickly using raster-scan hardware originally designed to efficiently display filled polygons on graphic terminals.) Each 2D bitmap may also be computed using the FFT-based method described in [Kav93]), whose complexity depends only on the size of the bitmap. This FFT method is also advantageous when the obstacles are originally input as bitmaps. In any case, experiments show that computing a 3D bitmap with a size on the order of $128 \times 128 \times 128$ takes a few seconds. The computation of the 3D bitmap(s) needed for collision checking is performed only once, prior to the learning phase.

Clearly, this technique is not yet practical for 3D workspaces, since it requires the generation of 6D bitmaps.

As mentioned above, there are many other ways of adjusting our general path planning method to a specific robot. For example, when placed in cluttered workspaces, robots of the type considered in this section yield C-spaces in which collision-free configurations form a tiny portion (typically a fraction of 1%) of the total space. Hence, a small ratio of the configurations which are randomly generated in the learning phase are collision-free. Most generated configurations are rejected by the collision-checking test. Several optimizations can be applied in this step. For example, we can draw the configuration coordinates in sequence from the base to the endpoint of the robot, and check a link for collision as soon as its location gets determined in order to discard configurations outside free C-space as early as possible.

However, too much specific tuning may not always be desirable, since it ultimately requires frequent changes in the implemented planner. At some point the gains in efficiency become too small and are no longer worth the burden of making the specific changes and keeping track of them.

5 Results with customized implementation

In this section we consider an implementation of the general method presented in Section 3, in which the local planner, the collision checker, and the distance function have been replaced by the specific ones described in Section 4. Actually to be precise, while collision checking with obstacles is done using the bitmap technique, self-collisions are detected analytically.

The planner is implemented in C and for the experiments reported here we used a DEC Alpha workstation (Model Flamingo). This machine is rated on the SPECMARKS benchmark with 126.0 SPECfp92, 74.3 SPECint92 and is running under DEC OSF/1.

We have tested our planner on a number of test scenes. Each such scene consists of a 2D workspace containing polygonal obstacles and a planar articulated robot whose links are line segments (see Figures 2 and 6). By no means does this reflect a limitation of the method. In particular, the specific local planner and collision checker of Section 4 apply as well to robots made of polygonal links (though several bitmaps may then be required). However, modeling links by line segments facilitates quick changes in the description of the robot and makes the graphic display of paths very easy.

The parameters given to our planner, which we consider in this section, are:

- $\mathtt{T_C}$, the time to be spent in the construction step;
- $\mathtt{T_E}$, the time to be spent in the expansion step;
- `maxdist`, the maximal distance between nodes that the local planner may try to connect;
- `eps`, the constant used to discretize local paths before collision checking;
- `maxneighbors`, the maximum number of calls of the local planner per node;
- $\mathtt{T_{RB\_expand}}$, the duration of the computation of a random-bounce walk performed during the expansion step (learning phase);
- $\mathtt{N_{RB\_query}}$, the maximum number of rbws allowed for connecting the start or goal configuration to the roadmap (query phase);
- $\mathtt{T_{RB\_query}}$, the duration of the computation of each of the rbws during the query phase.

(Notice that the last two parameters determines an upper bound on the time it takes to answer a query.)

For each test scene, we first input a set of configurations by hand, which we refer to as the test set. For a fixed $\mathtt{T_C}$ and $\mathtt{T_E}$, we then independently create many different roadmaps starting with different values of the random value generator. In the examples discussed here we only keep the largest connected component of the roadmap; other components, if any, are simply discarded. We then try to connect the same configuration to each of these roadmaps and we record the percentage of times our planner succeeds to make a connection in a prespecified amount of time (2.5 seconds). In this way, we believe that we present a quite realistic characterization of the performance of our planner. In particular, we ensure that the results do not reflect just a lucky run, or a bad one. We independently repeat the same experiment for a number of different times $\mathtt{T_C}$ and $\mathtt{T_E}$. For the other parameters described above, we choose fixed values throughout the experiments based on some preliminary experimental results. Notice that it is important to choose the configurations in the test set manually. For obvious reasons, a random generation similar to the one used during the learning phase tends to produce configurations that are very easily connected to the roadmap. Instead, proceeding manually allows us to select "interesting" configurations, for example configurations where the robot lies in narrow passages between workspace obstacles. It is unlikely that the random generator of the learning phase produced many such configurations.

We present results obtained with two representative scenes shown in Figures 2 (fixed-base robot) and 6 (free-base robot):

Fixed-base articulated robot. Figure 2 shows eight configurations forming the test set of a fixed-base articulated robot in a scene with several narrow gates.

Column 1 of the table in Figure 3 shows the total time, $\mathtt{T_L}$, spent in the learning phase. This time is broken into $\mathtt{T_C}$ and $\mathtt{T_E}$ in columns 2 and 3, with $\mathtt{T_E} = \mathtt{T_C}/2$. The values of the other parameters of the planner are: `maxdist` $= 0.4$, `eps` $= 0.01$ (for the interpretation of these two values note that the workspace is described as a unit square), `maxneighbors` $= 30$, $\mathtt{T_{RB\_expand}} = 0.01$ sec, $\mathtt{T_{RB\_query}} = 0.05$ sec, $\mathtt{N_{RB\_query}} = 45$.

For every row of the table in Figure 3 we separately generated 30 roadmaps, each with the indicated learning time. The roadmaps generated for different rows were also computed independently, that is, no roadmap in some row was reused to construct a larger one in following row.

Column 4 in Figure 3 gives the average number of nodes, over the 30 runs, in the largest roadmap component at the end of the learning phase. Columns 5 though 12 are labeled with the eight configurations $C_1, \ldots, C_8$ of Figure 2. They 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 (as defined by the parameters `maxdist`, `maxneighbors`, $\mathtt{T_{RB\_query}}$, and $\mathtt{N_{RB\_query}}$) was made per roadmap.

The table in Figure 3 shows that after a learning time of 60 seconds or more (rows 5, 6, and 7), all eight configurations of Figure 2 are successfully connected to the generated roadmaps with very few exceptions. These are all located in row 6, where configurations C_3, C_4 and C_7 were not connected to the produced roadmap, once out of the 30 trials of that row. Such exceptions are to be expected with a randomized technique.

Let us also note that actual timings for the connections of $C_1, \ldots, C_8$ to the roadmaps are very small: only a fraction of a second. This is shown in Figure 5 where we report the time it takes to connect the configurations to one of the 30 roadmaps produced, after learning times of 20, 30, 40, 50, 60, 70 and 80 seconds. Failure to connect to the largest component produced in less than 2.5 seconds is denoted by 'F'. In that table we report in column 4 the size of all the components produced with more than 10 nodes. It is easy to see that after a preprocessing time of 40 seconds, there is a clear difference in the size of the major component and the smaller ones. The latter contain only a small percentage of the total nodes and their presence does not affect path planning times.

Path planning will succeed between any two configurations that can be connected

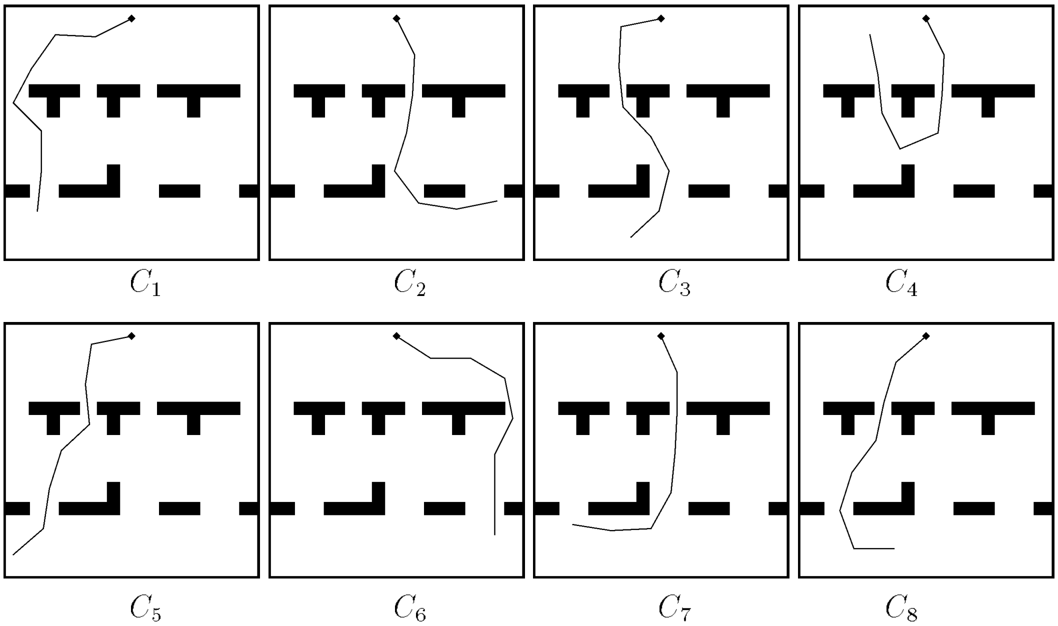

Figure 2: Scene 1, with 7-revolute-joint fixed-base robot.

T_L (sec)	T_C (sec)	T_E (sec)	Avg. nodes	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
20.4	13.3	7.0	975	100.0	26.7	36.7	13.3	40.0	96.7	26.7	43.3
30.0	19.5	10.5	1548	100.0	70.0	53.3	70.0	50.0	100.0	70.0	56.7
40.2	26.1	14.0	2102	100.0	80.0	76.7	80.0	80.0	100.0	80.0	83.3
50.1	32.5	17.5	2635	100.0	90.0	90.0	90.0	93.3	96.7	90.0	93.3
60.1	39.0	21.0	3147	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
70.4	45.8	24.6	3669	100.0	96.7	100.0	96.7	100.0	100.0	96.7	100.0
80.6	52.4	28.1	4061	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Figure 3: Results with customized planner for scene of Fig. 2 (with expansion).

T_L (sec)	T_C (sec)	T_E (sec)	Avg. nodes	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
20.2	20.2	0.0	947	100.0	10.0	23.3	10.0	26.7	73.3	10.0	23.3
30.3	30.3	0.0	1506	100.0	46.7	46.7	46.7	46.7	93.3	46.7	46.7
40.3	40.3	0.0	2150	100.0	73.3	76.7	73.3	76.7	100.0	73.3	76.7
50.3	50.3	0.0	2740	100.0	90.0	100.0	90.0	100.0	100.0	90.0	100.0
60.1	60.1	0.0	3211	100.0	90.0	100.0	90.0	100.0	93.3	90.0	100.0
70.3	70.3	0.0	3668	100.0	96.7	100.0	96.7	100.0	100.0	96.7	100.0
80.2	80.2	0.0	4103	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Figure 4: Results with customized planner for scene of Fig. 2 (no expansion).

$\mathtt{T_L}$ (sec)	$\mathtt{T_C}$ (sec)	$\mathtt{T_E}$ (sec)	Size of Components	Connection to roadmap time (sec)							
				C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
20.3	13.3	7.0	902,135,22	0.02	F	1.12	F	0.23	0.45	F	0.25
30.2	19.7	10.5	1607,144,12	0.00	0.02	F	0.40	F	0.55	0.00	F
40.4	26.3	14.1	2389,12	0.00	0.08	0.00	0.17	0.00	0.02	0.00	0.07
50.3	32.8	17.5	2879,43,15,10	0.02	0.02	0.00	0.17	0.07	0.00	0.02	0.05
60.4	39.3	21.1	3251,39,34	0.03	0.02	0.00	0.02	0.02	0.02	0.02	0.12
70.2	45.6	24.6	3717,50,43	0.02	0.02	0.00	0.00	0.02	0.00	0.02	0.02
80.2	52.1	28.1	4128,50,47	0.02	0.02	0.02	0.15	0.02	0.02	0.02	0.07

Figure 5: Timings for connecting configurations to the roadmap.

to the roadmaps produced. A simple breadth-first search algorithm typically takes less than 0.1 second to find a path between two nodes of the roadmaps in our examples. Thus, path planning between any two of $C_1, \ldots, C_8$ takes only a fraction of a second. This was the case for any two configurations we tried in the scene of Figure 2 and not only the eight configurations considered here.

Figure 4 shows the percentage of successful connections to roadmaps created with no expansion. The corresponding rows of the tables in Figures 3 and 4 report results obtained in the same learning time. We generated 30 independent roadmaps in each row in Figure 4. We again show the average number of nodes in their largest component (column 4) and the success rate when trying to connect $C_1, \ldots, C_8$ to these roadmaps. In general, the percentages of successful connections are lower in this table. The difference shows more clearly when the learning time is small. If we are interested in obtaining a solution to a path planning problem as fast as possible, it is thus better to spend part of the time allocated to the learning phase on the expansion step rather than spend it completely on the construction step. As mentioned above, the ratio $\mathtt{T_C}/\mathtt{T_E} = 2$ gives good results over a wide range of problems.

Free-base articulated robot. We have performed the same experiments for a free-base articulated robot (see Figure 6). The robot has a total of 7 dof: 2 for its free base and 5 for its revolute joints. The parameter values are the same as in the previous experiments.

Figures 7 and 8 show the results obtained with and without expansion, respectively. Again, in almost all cases, the percentage of successful connections to the roadmaps is greater with expansion than without (for the same total learning time). After a learning phase of 70 seconds, almost all configurations can be connected to

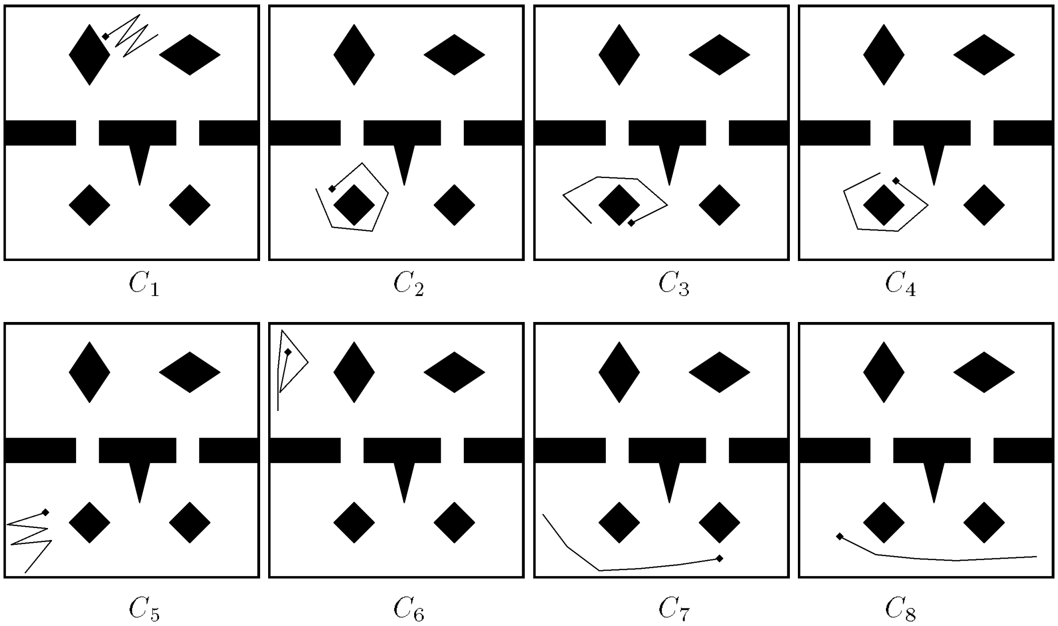

Figure 6: Scene 2, with 7-revolute-joint free-base robot.

$\texttt{T}_{\texttt{L}}$ (sec)	$\texttt{T}_{\texttt{C}}$ (sec)	$\texttt{T}_{\texttt{E}}$ (sec)	Avg. nodes	Success Rate (%)							
				C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
20.40	13.32	7.08	565	100.0	13.3	13.3	13.3	13.3	93.3	13.3	13.3
30.45	19.83	10.62	936	93.3	30.0	33.3	30.0	30.0	90.0	30.0	33.3
40.18	26.15	14.03	1571	100.0	60.0	60.0	60.0	60.0	100.0	60.0	60.0
50.20	32.63	17.57	2333	100.0	93.3	93.3	93.3	93.3	100.0	93.3	93.3
60.43	39.35	21.08	2850	100.0	93.3	93.3	93.3	93.3	100.0	93.3	93.3
70.33	45.80	24.53	3366	100.0	96.7	96.7	96.7	96.7	100.0	96.7	96.7
80.18	52.15	28.03	3837	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Figure 7: Results with customized planner for scene of Fig. 6 (with expansion).

$\texttt{T}_{\texttt{L}}$ (sec)	$\texttt{T}_{\texttt{C}}$ (sec)	$\texttt{T}_{\texttt{E}}$ (sec)	Avg. nodes	Success Rate (%)							
				C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
20.25	20.25	0.00	517	96.7	3.3	3.3	3.3	10.0	80.0	3.3	3.3
30.22	30.22	0.00	971	100.0	26.7	33.3	26.7	30.0	93.3	33.3	30.0
40.30	40.30	0.00	1348	100.0	33.3	33.3	33.3	33.3	100.0	33.3	33.3
50.06	50.05	0.02	2171	100.0	76.7	76.7	76.7	76.7	100.0	76.7	76.7
60.01	60.01	0.00	2632	100.0	80.0	80.0	80.0	80.0	100.0	80.0	80.0
70.28	70.28	0.00	3190	100.0	90.0	90.0	90.0	90.0	100.0	90.0	90.0
80.31	80.30	0.02	3836	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Figure 8: Results with customized planner for scene of Fig. 6 (no expansion).

the roadmaps produced. Actual timings for connecting $C_1, \ldots, C_8$ to the roadmaps are also in the order of a fraction of a second and path planning between any two of the eight shown configurations takes a fraction of a second.

6 Results with general implementation

The customized implementation used in the previous section solves efficiently path planning problems involving planar articulated robots. In this section we wish to demonstrate that the general implementation of the planner still gives very good results for a variety of examples.

The planner considered here is essentially an implementation of the method described in Section 3. Unlike the customized implementation, this implementation does not use any specific techniques for local path planning, collision checking, or distance computation. Hence, as described in Section 3, the local path constructed between any two configurations is the straight line segment joining them in C-space; the distance function D is the one defined by Equ. (1); and collision checking is done analytically, using routines from the PLAGEO library [Gie93]. We report here on experimentation conducted with articulated robots with 4 or 5 joints connected by polygonal links. However, as noted before, the same implementation is directly applicable to other holonomic robots, e.g., robots with polyhedral links moving in 3D workspaces.

The experiments were conducted on a Silicon Graphics Indigo^2 workstation with an R4400 processor running at 150 MHZ. This machine is rated on the SPECMARKS benchmark with 96.5 SPECfp92 and 90.4 SPECint92. It is comparable to the machine we used for the results in the previous section.

We present results obtained with two representative examples. In scene 1, shown in Figure 9, we have a 4-dof robot with three revolute joints and one prismatic joint (indicated by the double arrow). Scene 2, shown in Figure 10, is a slightly more difficult one, with a five-revolute-joint robot and narrow areas in the workspace. For most existing planners, motion planning problems in both these scenes would be challenging ones. Still, they are considerably easier than in the scenes of Section 5, due to the relatively low number of dofs of the two robots, and the presence of only few tight areas in the workspaces of the robots.

The experiments conducted with these two test scenes are similar, though somewhat simpler, than those in Section 5. For each scene, we consider only two "difficult"

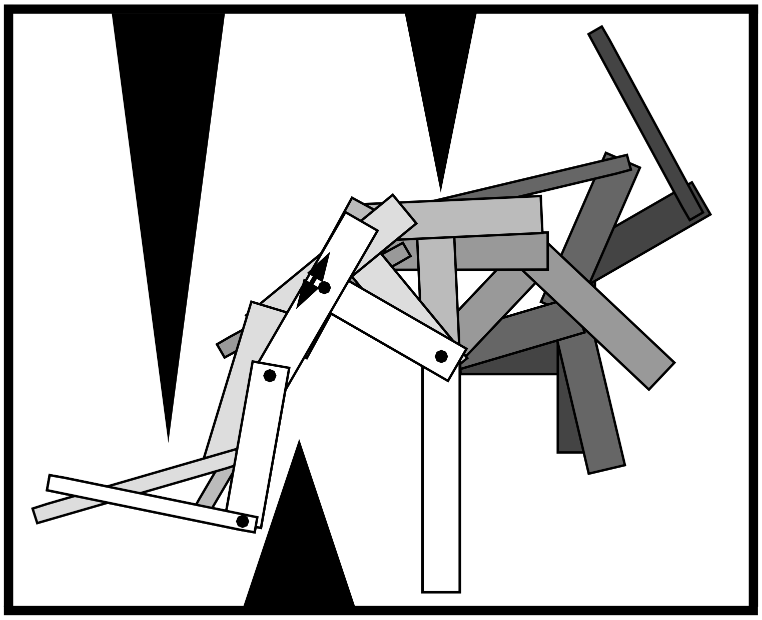

Figure 9: Scene 1, with four-dof robot.

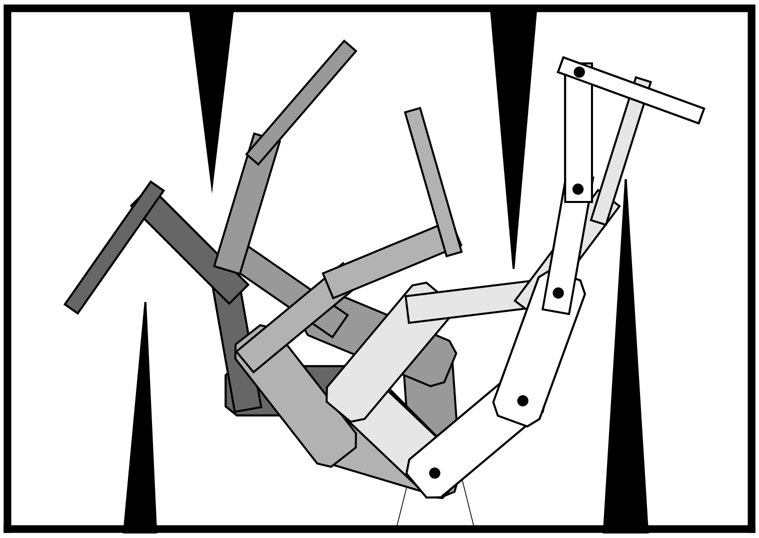

Figure 10: Scene 2, with five-dof robot.

$\mathtt{T_L}$ (sec)	$\mathtt{T_C}$ (sec)	$\mathtt{T_E}$ (sec)	Success rate in Scene 1 (%)	Success rate in Scene 2 (%)
5	3.33	1.67	50	37
10	6.66	3.34	80	87
15	10	5	97	93
20	13.33	6.67	100	100

Figure 11: Results with general planner for scenes of Fig. 9 and 10.

configurations s and g. Then, for a fixed construction time $\mathtt{T_C}$ and expansion time $\mathtt{T_E}$ (hence, a fixed learning time $\mathtt{T_L}$), we independently create 30 roadmaps. For each of these roadmaps we only consider its main connected component and we test whether the query with configurations (s, g) succeeds within 2.3 seconds. In other words, we test whether *both* s and g can be quickly connected to the main connected component of the roadmap with the method described in Section 3.2. We repeat this experiment for a number of different construction times $\mathtt{T_C}$ and expansion times $\mathtt{T_E}$, with $\mathtt{T_E} = \mathtt{T_C}/2$). For each such pair of times we report the success rate in percent of the query phase.

The other parameters have the following fixed values, which are almost the same as in the experimentation reported in the previous section: `maxdist` = 0.5, `eps` = 0.01, `maxneighbors` = 30, $\mathtt{T_{RB\_expand}} = 0.01$, $\mathtt{T_{RB\_query}} = 0.05$ sec, and $\mathtt{N_{RB\_query}} = 45$. Again, for the interpretation of the values for `maxdist` and `eps`, note that we scaled the two scenes in a way that the workspace obstacles just fit into the unit square.

In both Figures 9 and 10 the start configuration s is shown in dark grey, and the goal configuration R in white. In each figure, several robot configurations along a path solving the query are displayed using various grey levels. The results of the experiments described above are given in Figure 11. We see that the query in scene 1 is solved in all 30 cases after having learned for 20 seconds. Learning for 10 seconds though suffices to successfully answer the query in 80% of the cases. In scene 2 we observe a similar behavior.

These results show that the general implementation is able to efficiently solve rather complicated planning problems. However, when applied to problems involving more dofs, like those in the previous section, the learning times required to build good roadmaps are much longer. For example, experiments indicated that about 30 minutes of learning are required in order to obtain roadmaps that capture well the free C-space connectivity of the scene shown in Figure 2. Figure 12 reports some experimental results. As in Section 5, we show the percentage of times that our

T_L (min)	T_C (min)	T_E (min)	Success Rate (%) C_1	C_4	C_7	C_8
5	3.3	1.7	76.7	10.0	23.3	26.7
10	6.7	3.3	96.7	66.7	70.0	53.3
15	10	5.0	96.7	73.3	66.7	80.0
20	13.3	6.7	100.0	93.3	83.3	93.3
25	16.7	8.3	100.0	96.7	96.7	100.0
30	20	10	100.0	100.0	100.0	100.0

Figure 12: Results with general planner for scene of Fig. 2 (with expansion).

planner succeeds to connect the specified configurations to the roadmap, over many independently constructed roadmaps, for different learning times. In such difficult cases, clearly, customization is desirable, if not necessary.

7 Conclusion

We have described a two-phase method to solve robot motion planning problems in static workspaces. In the learning phase, the method constructs a probabilistic roadmap as a collection of configurations randomly selected across 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 learning 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 is key to solving difficult queries.

The method is general and can be applied to virtually any type of holonomic robot. Furthermore, it can be easily customized to run more efficiently on some family of problems. Customization consists of replacing general components of the method, such as the local planner, by more specific ones fitting better the characteristics of the considered scenes. In this paper we have reported on the application of the method to planar articulated robots. We have described techniques to customize the method to such robots and we have presented experimental results with both a general and a customized implementation of the method. The customized implementation can solve very difficult path planning queries involving many-dof robots in a fraction of a second, after a learning time of a few dozen seconds. The general implementation efficiently solves less difficult, but still challenging problems, demonstrating the power of our method.

In [KL94a, KL94b, OŠ94] prior versions of the method have been applied to a great variety of holonomic robots including planar and spatial articulated robots with revolute, prismatic, and/or spherical joints, fixed or free base, and single or multiple kinematic chains. In [Šve93, ŠO94] a variation of the method (essentially one with a different general local planner) was also run successfully on examples involving nonholonomic car-like robots.

Experimental results show that our method can efficiently solve problems which are beyond the capabilities of other existing methods. For example, for planar articulated robots with many dofs, the customized implementation of Section 5 is much more consistent than the Randomized Path Planner (RPP) of [BL91]. Indeed, the latter can be very fast on some difficult problems, but it may also take prohibitive time on some others. We have not observed such disparity with our roadmap method. Moreover, after sufficient learning (usually on the order of a few dozen seconds), the probabilistic roadmap method answers queries considerably faster than RPP. However, when the learning time is included in the planning time, RPP is faster on many problems, since it does not perform any substantial precomputation.

An important question is how our method scales up when we consider scenes with more complicated geometry, since the cost of collision checking can be expected to increase. First, let us note that in 2D workspaces the effect is likely to be limited if the bitmap collision-checking technique of Section 4 is used. Indeed, once bitmaps have been precomputed, collision checking is a constant-time operation; and the cost of computing bitmaps using the FFT-based technique described in [Kav93] only depends on the resolution (i.e., the size) of these bitmaps. However, more complicated geometry may require increasing bitmap resolution in order to represent geometric details with desired accuracy. With 3D workspaces the situation is completely different, since we can no longer use the bitmap technique. Our experiments in 3D workspaces reported in [KL94b] show that the higher cost of collision checking mainly increases the duration of the learning phase. Indeed, in the query phase, collision checking is needed only to connect the start and goal configurations to the roadmap. The results in [KL94b] also show that the duration of the learning phase remains quite reasonable (on the order of minutes), but they were obtained with simple 3D geometry (for example, the robot links were line segments). For more complicated geometries, the use of an iterative collision checker, like the one in [Qui93], will be advantageous. The collision checker in [Qui93] considers sucessive approximations of the objects and its running time, on the average, does not depend much on the geometric complexity of the scenes. RPP is another planner that heavily relies on collision checking. For long we ran RPP on geometrically simple problems; but, recently, we used it to automatically animate graphic 3D scenes of complex geometry [KKKL94] using the above iterative collision checker. We observed no dramatic slowdown of the planner.

A challenging research goal would now 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? Answering this question would be useful to apply our method to scenes subject to small incremental changes. Such changes occur in many manufacturing (e.g., assembly) cells; while most of the geometry of such a cell is permanent and stationary, a few objects (e.g., fixtures) are added or removed between any two consecutive manufacturing operations. Similar incremental changes also occur in automatic graphic animation. A second question is: How should the learning and query phase be modified if some obstacles are moving along known trajectories? An answer to this question might consist of applying our roadmap method in the configuration$\times$time space of the robot [Lat91]. The roadmap would then have to be built as a directed graph, since local paths between any two nodes must monotonically progress along the time axis, with possibly additional constraints on their slope and curvature to reflect bounds on the robot's velocity and acceleration.

References

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

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

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

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

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

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

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

[FT87] B. Faverjon and P. Tournassoud. A local approach for path planning of manipulators with a high number of degrees of freedom. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1152–1159, Raleigh, NC, 1987.

[FT90] B. Faverjon and P. Tournassoud. A practical approach to motion planning for manipulators with many degrees of freedom. In *Robotics Research 5*, H. Miura and S. Arimoto (Eds.), pages 65–73, MIT Press, Cambridge, MA, 1990.

[GG92] K. Gupta and Z. Gou. Sequential search with backtracking. In *Proc. of IEEE Int. Conf. Robotics and Automation*, pages 2328–2333, Nice, France, 1992.

[Gie93] G.-J. Giezeman. *PlaGeo—A Library for Planar Geometry.* Tech. Rep., Dept. Comput. Sci., Utrecht Univ., Utrecht, The Netherlands, August 1993.

[GMKL92] L. Graux, P. Millies, P.L. Kociemba, and B. Langlois. Integration of a path generation algorithm into off-line programming of airbus panels. *Aerospace Automated Fastening Conf. and Exp.*, SAE Tech. Paper 922404, October 1992.

[GZ94] K. Gupta and X. Zhu. 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, 1994.

[HST94] Th. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom - random reflections at c-space obstacles. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2138–2145, San Diego, CA, 1994.

[Kav93] L. Kavraki. Computation of configuration-space obstacles using the fast fourier transform. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 255–261, Atlanta, GA, 1993. To appear in *IEEE Tr. Robotics and Automation.*

[KL93] L. Kavraki and J.-C. Latombe. *Randomized Preprocessing of Configuration Space for Fast Path Planning.* Tech. Rep. STAN-CS-93-1490, Dept. Comput. Sci., Stanford Univ., Stanford, CA, September 1993.

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

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

[KKKL94] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe. Planning Motions with Intentions. In *Proc. of SIGGRAPH'94*, pages 395-408, FL, 1994.

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

[Lat91] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[LRDG90] J. Lengyel, M. Reichert, B.R. Donald, and P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. In *Proc. SIGGRAPH'90*, pages 327–335, Dallas, TX, 1990.

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

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

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

[Mas92] J. Mastwijk. *Motion Planning Using Potential Field Methods.* Master Thesis, Dept. Comput. Sci., Utrecht Univ., Utrecht, The Netherlands, August 1992.

[OY82] C. Ó'Dúnlaing and C.K. Yap. A retraction method for planning the motion of a disc. *J. of Algorithms*, 6:104-111, 1982.

[OŠ94] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In *Proc. Workshop Algorithmic Foundations of Robotics*, San Francisco, CA, 1994 (to appear).

[Ove92] M. Overmars. *A Random Approach to Motion Planning.* Tech. Rep. RUU-CS-92-32, Dept. Comput. Sci., Utrecht Univ., Utrecht, The Netherlands, October 1992.

[Qui93] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 3324-3330, San Diego, CA, 1994.

[ŠO94] P. Švestka and M. Overmars. *Motion Planning for Car-Like Robots, Using a Probabilistic Learning Approach.* Tech. Rep. RUU-CS-94-33, Dept. Comput. Sci., Utrecht Univ., Utrecht, The Netherlands, May 1994.

[Šve93] P. Švestka. *A Probabilistic Approach to Motion Planning for Car-Like Robots.* Tech. Rep. RUU-CS-93-18, Dept. Comput. Sci., Utrecht Univ., Utrecht, The Netherlands, April 1993.

[ZG93] X. Zhu and K. Gupta. *On Local Minima and Random Search in Robot Motion Planning.* Tech. Rep., Simon Fraser Univ., Burbany, BC, Canada, 1993.