A Paradigm for Probabilistic Path Planning

Mark H. Overmars, Petr Švestka
Department of Computer Science, Utrecht University
P.O.Box 80.089, 3508 TB Utrecht, the Netherlands
e-mail: markov@cs.ruu.nl, petr@cs.ruu.nl

March 4, 1996

Contents

1 Introduction

Recently there has been a renewed interest in developing heuristic, but practical motion planners. This was motivated by the fact that exact methods, although often practical for simple problems involving holonomic robots with few degrees of freedom, fail to solve efficiently more challenging (but practical) problems involving high dof robots or robots with nonholonomic constraints.

Among the most successful planners is RPP ([2]), a potential field based method that uses Brownian motion for escaping from local minima. It has successfully been applied to robots with many degrees of freedom (up to 31), and it has been used in practice [15] to plan motions for riveting operations on plane fuselages. The results where good. In [13] the method is used for automatically generating graphic animations of humane figures, modelled with 62 dofs. In [16] the probabilistic convergence of the used Brownian motions is proven, and a finite estimate of the expected complexity is given.

Genetic algorithms are utilized in [1] for guiding path searches in high dimensional configuration spaces, and, for certain problems, this algorithm also proves to be very efficient. A heuristic learning approach using cell-decomposition of the configuration space is presented in [5]. A r_n array is used, where n is the number of dofs and r is the number of intervals discretizing the range of each dof, to accumulate probabilities of successfully moving between neighboring cells (with a potential field based local method) without getting stuck in a local minimum. For robots with up to 6 dofs fairly good results where obtained. The size of r_n however clearly becomes impractical for large n. In [14] a potential field approach is described that uses heuristics to limit the configuration space portion that is explored. Finally we mention the work in [6], where paths for high dof robots are planned using random reflections at configuration space obstacles.

In this chapter we describe a probabilistic paradigm to the motion planning problem, which proves to be very time-efficient for a great variety of robots, including high dof articulated robots and robots with various nonholonomic constraints, in constrained environments. An advantage over the above mentioned methods is its generality. There are only a few components that are robot specific, and these are, as will be pointed out in this chapter, easy to define/implement. Furthermore, it is a learning approach, that is, it builds data-structures that, once constructed, can be used for retrieving arbitrary paths quasi-instantaneously. Potential field methods inherently do not have this property, due to the fact that the potential field always depends on the goal configuration of a particular path planning problem. Also the presented method is probabilistically complete.

In the probabilistic paradigm, there is a (robot independent) global planner, and a (robot specific) local planner. The global planner itself consists of two phases. In the learning phase the probabilistic global planner incrementally constructs a roadmap, which is stored as a graph with nodes corresponding to probabilistically chosen collision-free configurations, and edges to simple feasible paths, or local paths. The aim is to obtain a road-map that effectively captures the connectivity of the free configuration space (or free *C-space*). In the query phase, the roadmap is used for retrieving feasible paths connecting given start and goal configurations of the robot. That is, paths that are collision free and that respect the robots constraints. A local planner is used for computing the local paths, corresponding to the edges in the graph. This is a deterministic planner that, given two configurations, efficiently tries to construct a path connecting them. This path always is feasible in absence of obstacles. It is stored as an edge in the graph if and only if it is collision free. A proper choice of the local

planner guarantees probabilistic completeness of the resulting global planner.

Planners based on the paradigm have been implemented for free-flying robots ([21]), (high dof) articulated robots ([10],[22],[12]), and various types of nonholonomic robots ([31], [33]). Extensions of the method have been used for solving multi-robot path planning problems ([32]) and (simple) problems involving movable obstacles ([34]). In [11] a theoretical bases is initiated for explaining the success of planners based on the probabilistic paradigm. A local planner L being given, the notion of *reachable sets* of introduced. The reachable set of a configuration c consists of the C-space region that is reachable from c by L (that is, without collisions). Under realistic assumptions about the volume of the smallest such set, the authors provide expected complexity bounds for the method.

In this chapter an overview is given on different applications of the probabilistic paradigm. It is organized as follows: In Section 2 the paradigm is described in its general form. In the following two sections it is applied to specific robot types, i.e., to holonomic robots (free-flying and articulated) in Section 3, and to nonholonomic mobile robots in Section 4. In both sections the robot specific components of the algorithm are defined, and experimental results obtained with implementations in C++ are presented. Section 5 is of a more theoretical nature. Aspects regarding probabilistic completeness of the method are discussed, and proofs of probabilistic completeness are given for the planners described in this chapter. In Section 6 an extension of the paradigm for solving multi-robot path planning problems is described, and experimental results are given for problems involving multiple carlike robots. Some final comments are made in Section 7.

2 The probabilistic paradigm

The probabilistic learning paradigm can be described in general terms, without focussing on any specific robot type. The idea is that during the *learning phase* a data structure is incrementally constructed in a probabilistic way, and that this data structure is later, in the *query phase*, used for solving individual motion planning problems.

The data-structure constructed during the learning phase is an undirected graph $G = (V, E)$, where the nodes V are probabilistically generated free configurations and the edges E correspond to (simple) feasible paths. These simple paths, which we refer to as *local paths*, are computed by a *local planner*, which should be a very simple but fast and deterministic motion planner. If the local planner is chosen properly (see Section 5), then one can prove probabilistic completeness of the global planner.

In the query phase, given a start configuration s and a goal configuration g, we try to connect s and g to suitable nodes $\tilde{s}$ and $\tilde{g}$ in V. Then we perform a graph search to find a sequence of edges in E connecting $\tilde{s}$ to $\tilde{g}$, and we transform this sequence into a feasible path in the (free) C-space. So the paths generated in the query phase (which is described in detail later) are basically just concatenations of local paths, and therefore the properties of these "global paths" are induced by the local planner. This makes our approach a flexible one.

2.1 The learning phase

We assume that we are dealing with a robot $\mathcal{A}$, and that L is a local planner that computes paths feasible for $\mathcal{A}$. As mentioned above, in the learning phase a probabilistic roadmap is constructed, and stored in an undirected graph $G = (V, E)$. The construction of the roadmap is performed incrementally in a probabilistic way. Repeatedly a random free configuration

c is generated and added to V. Heuristics however are used for generating more nodes in "difficult" areas of the free C-space. We try to connect each generated node c to the graph by adding a number of edges (c, n) to E, such that the local planner can connect from c to n.

This edge adding is done as follows: First, a set $N(c)$ of neighbors is chosen from N. This set consists of nodes lying within a certain distance from c, with respect to some metric D. Then, in order of increasing distance from c, we pick nodes from $N(c)$. 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. The motivation for preventing cycles is that no query would ever succeed *thanks to* an edge that is part of a cycle. Hence, adding an edge that creates a cycle can impossibly improve the planners performance in the query phase.

A price to be paid for disallowing cycles in the graph is that in the query phase often unnecessarily long paths will be obtained. Suppose that a and b are two configurations that can easily be connected by some short feasible path. Due to the probabilistic nature of the learning algorithm, it is very well possible that, at some point, a and b get connected by some very long path. Obtaining a shorter connection between a and b would require the introduction of a cycle in the graph, which we prevent. So, for any pair of nodes, the first graph path connecting them blocks other possibilities.

There are a number of ways for dealing with this problem. One possibility is to apply an edge adding method which does allow cycles in the graph ([22]). These methods however have the disadvantage that they slow down the learning algorithm, due to the fact that the adding of a node requires more executions of the local method to be performed. Another possibility is to build a forest as described above, but, before using the graph for queries, "smoothing" the graph by adding certain edges which create cycles. Some experiments that we have done indicated that smoothing the graph for just a few seconds significantly reduces the path lengths in the query phase. Finally, it is possible to apply some smoothing techniques on the paths constructed in the query phase. We briefly describe a simple but efficient and general probabilistic path smoothing technique in Section 2.4.

Let $\mathcal{C}$ be the C-space of the robot. To describe the learning algorithm formally, we need the following :

- A symmetric function $L_d \in \mathcal{C} \times \mathcal{C} \rightarrow \mathit{boolean}$, that returns whether the local planner can compute a feasible path for $\mathcal{A}$ between its two argument-configurations.
- A function $D \in \mathcal{C} \times \mathcal{C} \rightarrow \mathbf{R}^+$. It defines the metric[1] used, and should give a suitable notion of distance for arbitrary pairs of configurations, taking the properties of the robot $\mathcal{A}$ into account. We assume that D is symmetric.

The algorithm can now be described as follows:

The learning algorithm:

(1) $V = \emptyset$, $E = \emptyset$
(2) **loop**
(3) $\quad c =$ a "randomly" chosen free configuration
(4) $\quad V = V \cup \{c\}$
(5) $\quad N(c) =$ a set of neighbors of c chosen from V

[1] By metric we simply mean a function of type $\mathcal{C} \times \mathcal{C} \rightarrow \mathbf{R}^+$, without any restrictions.

(6) **forall** $n \in N_c$, in order of increasing $D(c,n)$ **do**
(7) **if** $\neg connected(c,n) \wedge L_d(c,n)$ **then** $E = E \cup \{(c,n)\}$

The learning method, a described above, leaves a number of choices to be made: A local planner must be chosen, a metric must be defined, and it must be defined what the neighbors of a node are. Furthermore, heuristics for generating more nodes in interesting C-space areas should be defined. Some choices must be left open as long as we do not focus on a particular robot type, but certain global remarks can be made here.

Local planner One of the crucial ingredients in the learning phase is the local planner. As mentioned before, the local planner must compute paths which are *feasible* for $\mathcal{A}$. Furthermore, the local planner should be deterministic. Otherwise the existence of a path in G between two nodes a and b does not guarantee that a feasible path in C-space connecting a and b can be reconstructed in the query phase. Another requirement is that the local planner always terminates (some potential field methods do not have this property). Finally, the local planner should guarantee probabilistic completeness of the learning algorithm. In Section 5 we give sufficient properties.

There are still many possible choices for such an algorithm. On one hand one could take a very powerful planner. Such a planner would very often succeed in finding a feasible path when one exists, and, hence, relatively few nodes would be required in order to obtain a graph which captures the connectivity of the free C-space well. Such a local planner would (probably) be slow, but one could hope that this is compensated by the fact that only a few executions of the planner need to be performed. On the other hand, one could choose a very simple and fast algorithm that is much less successful. In this case many more nodes will have to be added in order to obtain a reasonable graph, which means that many more executions of the local planner will be required. But this might be compensated by the fact that each execution is very cheap. So it is clear that there is a trade-off, and it is not trivial to make a smart choice here.

We have guided the choice of our local planners by experiments ([30],[21]). These clearly indicated that very fast (and, hence, not very powerful) local planners lead to the best performance of the learning algorithm.

Neighbors and edge adding methods Another important choice to be made is that of the neighbors $N(c)$ of a (new) node c. As is the case for the choice of the local planner, the definition of $N(c)$ has large impact on the performance of the learning algorithm. Reasons for this are that the choice of the neighbors strongly influences the overall structure of the graph, and that, regardless of how the local planner is exactly defined, the executions of the local planner are by far the most time-consuming operations of the learning algorithm (due to the collision tests that must be performed).

So it is clear that executions of the local planner that do not effectively extend the knowledge stored in the roadmap should be avoided as much as possible. Firstly, as mentioned before, attempts to connect to nodes which are already in c's connected component are useless. For this reason the learning algorithm builds a forest. Secondly, local planner executions which fail add no knowledge to the roadmap. To avoid too many local planner failures we only submit pairs of configurations whose relative distance (with

respect to D) is relatively small, that is, less than some constant threshold *maxdist*. Thus:

$$N(c) \subset \{\tilde{c} \in V | D(c, \tilde{c}) \leq maxdist\} \tag{1}$$

This criterion still leaves many possibilities open, regarding the actual choice for $N(c)$. We have decided on taking all nodes within distance *maxdist* as neighbors. Experiments with various definitions for $N(c)$ on a wide range of problems lead to this choice.

Hence, according to the algorithm outline given above, we try to connect to all "nearby" nodes of c, in order of increasing distance D, but we skip those nodes which are already in c's connected component at the time that the connection is to be attempted. 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). We refer to the described edge adding method as the *forest method*.

Distance We have seen that a distance function D is used for choosing and sorting the neighbors $N(c)$ of a new node c. It should be defined in such a way that $D(a, b)$ (for arbitrary a and b) somehow reflects the chance that the local planner will *fail* to compute a feasible path from a to b. For example, given two configurations a and b, a possibility is to define $D(a, b)$ as the size of the sweep volume (in the workspace) constructed when the local planner computes a path connecting a to b, in the absence of obstacles. In this way each local planner L induces its own metric, which reflects the described "failure-chance" very well. In fact, if the obstacles were randomly distributed points, then this definition would reflect the local planner's failure chance exactly. In the general case however, exact computations of the described sweep-volumes tend to be rather expensive, and in practice it turns out that certain rough but cheap to evaluate approximations of the sweep volumes are to be preferred.

Node adding heuristics If the number of nodes generated during the learning phase is large enough, the set V gives a fairly uniform covering of the free C-space. In easy cases, for example for holonomic robots with few degrees of freedom (say not more than 4), G is then well connected. But in more complicated cases where free C-space is actually connected, G tends to remain disconnected for a long time in certain narrow (and hence difficult) areas of the free C-space.

Due to the probabilistic completeness of the method, we are sure that eventually G will grasp the connectivity of the free space, but to prevent exorbitant running times, it is wise to guide the node generation by heuristics which create higher node densities in the difficult areas. To identify these, there are a number of possibilities.

In some cases, one can use the geometry of the workspace obstacles. For example, for carlike robots adding (extra) configurations which correspond to placements of the robot "parallel" to obstacle edges and "around" convex obstacle corners boosts the performance of the learning phase significantly.

A more general criterion is to use the (run-time) structure of the roadmap G. Given a node $c \in V$, one can count the number of nodes of V 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 c lies in a difficult area. Another possibility is to look at the distance 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).

Alternatively, rather than using the structure of the obstacles or the roadmap to identify difficult regions, one can look at the run-time 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.

2.2 The query phase

During the query phase, paths are to be found between arbitrary start and goal configurations, using the graph G computed in the learning phase. The idea is that, given a start configuration s and a goal configuration g, we try to find feasible paths P_s and P_g, such that P_s connects s to a graph node $\tilde{s}$, and P_g connects g to a graph node $\tilde{g}$, with $\tilde{s}$ graph-connected to $\tilde{g}$ (that is, they lie in the same connected component of G). If this succeeds, we perform a graph search to obtain a path P_G in G connecting $\tilde{s}$ to $\tilde{g}$. A feasible path (in C-space) from s to g is then constructed by concatenating P_s, the subpaths computed by the local planner when applied to pairs of consecutive nodes in P_G, and P_g reversed. Otherwise, the query fails. The queries should preferably terminate 'instantaneously', so no expensive algorithm is allowed for computing P_s and P_g.

For finding the nodes $\tilde{s}$ and $\tilde{g}$ we use the function $query\_mapping \in \mathcal{C} \times \mathcal{C} \rightarrow V \times V$, defined as follows :

$$query\_mapping(a,b) = (\tilde{a},\tilde{b}), \text{ such that } \tilde{a} \text{ and } \tilde{b} \text{ are connected, and}$$
$$D(a,\tilde{a}) + D(b,\tilde{b}) = \text{MIN}_{(x,y)\in W} : D(a,x) + D(y,b)$$

$$\text{where } W = \{(x,y) \in V \times V \,|\, connected(x,y)\}$$

So $query\_mapping(a,b)$ returns the pair of connected graph nodes $(\tilde{a},\tilde{b})$ which minimize the total distance from a to $\tilde{a}$ and from b to $\tilde{b}$. We will refer to $\tilde{a}$ as a's graph retraction, and to $\tilde{b}$ as b's graph retraction.

The most straightforward way for performing a query with start configuration s and goal configuration g is to compute $(\tilde{s},\tilde{g}) = query\_mapping(s,g)$, and to try to connect with the local planner from s to $\tilde{s}$ and from $\tilde{g}$ to g. The local planner though typically is a rather weak planner, and, in unlucky cases, it may fail to find the connections even if the graph captures the connectivity of free C-space well.

Experiments with different robot types indicated that simple probabilistic methods that (repeatedly) perform short random walks from s and g, and try to connect to the graph retractions of the end-points of those walks with the local planner, achieve significantly better results. These random walks should be aimed at maneuvering the robot out of narrow C-space areas (that is, areas where the robot is tightly surrounded by obstacles), and hereby improving the chances for the local planner to succeed. For holonomic robots very good performance is obtained by what we refer to as the *random bounce walk* (see also [12]). The idea is that repeatedly a random direction (in C-space) is chosen, and the robot is moved in this direction

until a collision occurs (or time runs out). When a collision occurs, a new random direction is chosen. This method performs much better than for example pure Brownian motion in C-space. For nonholonomic robots walks of a similar nature can be performed, but care must of course be taken to respect the nonholonomic constraints.

2.3 Using a directed graph

In the probabilistic learning paradigm, as described in the previous section, the computed roadmaps are stored in undirected graphs. For many motion planning problems this is sufficient, and it appears that the method is easier and more efficient to implement when based on undirected graphs. For example, motion planning problems involving free-flying robots, articulated robots, and general carlike robots can all be dealt with using undirected underlying graphs. There are however motion planning problems for which undirected underlying graphs not sufficient, and directed ones are required instead. For example, problems involving forward carlike robots require directed underlying graphs.

The existence of an edge (a, b) in the underlying graph G corresponds to the statement that the local planner can compute a feasible path from a to b. If though G is undirected, then the edge contains no information about the direction in which the local planner can compute the path, and, hence, it must correspond to the statement that the local planner can compute both a feasible path from a to b, as well as one from b to a. So an edge (a, b) can be added only if the local planner succeeds in both directions. Doing so, useful information might be thrown away. This will happen in those cases where the local planner is successful in exactly one direction, and the fact that it has successfully computed a feasible path will not be stored. If however the local planner is *symmetric*, which means that it succeeds for say (a, b) whenever it succeeds for (b, a), then obviously this problem will never occur. So if the local planner is symmetric, the underlying graph can be undirected, and if it is not symmetric, then it is better to use a directed graph.

Whether it is possible to implement (good) local planners which are symmetric, depends on the properties of the robot $\mathcal{A}$, defined by the constraints imposed on it.

Definition 1 *A robot $\mathcal{A}$ has the* reversibility property **iff** *any feasible path for $\mathcal{A}$ remains feasible when reversed.*

Holonomic robots and general carlike robots are two examples of robot types which possess the reversibility property, while for example forward carlike robots do not possess the reversibility property. In terms of control theory, a robot has the reversibility property if its control system is symmetric. That is, it can attain a velocity v (in C-space) if and only if it can also attain velocity $-v$.

Clearly, if $\mathcal{A}$ has the reversibility property, then any local planner L that computes feasible paths for $\mathcal{A}$ can be made symmetric in a trivial way, by reversing computed paths when necessary. So this implies that if the robot has the reversibility property, then an undirected graph can be used for storing the local paths, and otherwise a directed graph is required.

For directed graphs it is less straightforward to omit the adding of redundant edges than was the case for undirected graphs. We refer to [30] and [31] for discussions on this topic, and sensitive strategies for edge adding for directed underlying graphs.

2.4 Smoothing the paths

Paths computed in the query phase can be quite ugly and unnecessarily long. This is due to the probabilistic nature of the algorithm, and to the fact that edge-creating edges are never added.

To improve this, one can apply some path smoothing techniques on these 'ugly' paths. The smoothing routine that we use is very simple. It repeatedly picks a pair of random configurations (c_1, c_2) on the "to be smoothed" path P_C, tries to connect these with a feasible path Q_{new} using the local planner. If this succeeds and Q_{new} is shorter than the path segment Q_{old} in P_C from c_1 to c_2, then it replaces Q_{old} by Q_{new} (in P_C). So basically, randomly picked segments of the path are replaced, when possible, by shorter ones, computed by the local planner. The longer this is done, the shorter (and nicer) the path gets. Typically, this method smoothes a path very well in less than a second for low dof robots, and in a few seconds for high dof robots.

Still one can argue that this is too much for a query. In that case one must either accept the ugly paths, or use a more expensive edge adding method which builds graphs containing loops. This will result in a slowdown of the learning phase, but the gain is that the paths which are (directly) retrieved in the query phase will be shorter.

3 Application to holonomic robots, and experimental results

In this section an application of the probabilistic paradigm to two types of holonomic robots is described: free-flying robots and articulated robots.

We consider here only planar holonomic robots. A free-flying robot is represented as a polygon that can rotate and translate freely in the plane among a set of polygonal obstacles. Its C-space is represented by $R^2 \times [0, 2\pi[$. A planar articulated robot $\mathcal{A}$ consists of n *links* $L_1, \ldots, L_n$, which are some solid planar bodies (we use polygons), connected to each other by $n-1$ *joints* $J_2, \ldots, J_n$. Furthermore, the first link L_1 is connected to some *base point* in the workspace by a joint J_1. Each joint can be either a *prismatic joint*, or a *revolute joint*. If J_i is a prismatic joint, then link L_i can translate along some vector, which is fixed to link L_{i-1} (or to the workspace, if $i = 1$), and if J_i is a revolute joint, then link L_i can rotate around some point which is fixed to link L_{i-1} (or to the workspace, if $i = 1$). The range of the possible translations or rotations of each link L_i is constrained by J_i's *joint bounds*, consisting of a lower bound low_i and an upper bound up_i. The C-space of a n-linked planar articulated robot can, hence, be represented by $[low_1, up_1] \times [low_2, up_2] \times \ldots \times [low_n, up_n]$. In the scenes we show, the revolute joints are indicated by small black discs, and the prismatic joints by small black discs with double arrows.

Since holonomic robots have the reversibility property, it is feasible to use undirected graphs for storing the roadmaps. Some of the (robot specific) details, left open in the discussion of the general method, must be specified.

3.1 Filling in the details

The local planner: A very general local planner exists, that is directly applicable to all holonomic robots. Given two configurations, it connects them by a straight line segment in C-space and checks this line segment for collision and joint limits (if any). We refer to this planner as *the general holonomic local planner*. Collision checking can be done

as follows: First, discretize the line segment 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 ϵ away from its position when the robot is at configuration c_{i+1} (ϵ is an input positive constant). Then, for each configuration c_i, test whether the robot, when positioned at c_i and "grown" by ϵ, is collision-free. If none of the m configurations yield collision, conclude that the path is collision-free.

The metric: The distance between two configurations a and b is defined as the length (in C-space) of the local path connecting a and b, but scaled in the various C-space dimensions appropriately, in order to reflect the local planners failure chance reasonably. For example, in the case of of a long and thin free flying robot, small variations in orientation (that is, variations in the third dimension) correspond to motions sweeping relatively large volumes in the workspace, and should hence be reflected by large distances, while, on the other hand, for disc-like robots they should be reflected by small distances.

The random walks in the query phase: Section 2.2 described a general scheme for solving a query using a graph constructed in the learning phase. Multiple random walks were performed from the query configurations s and g, aimed at connecting the endpoints of these walks to their graph retractions with the local planner. Remains to define the specific random walks. For holonomic robots, a random bounce walk 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.

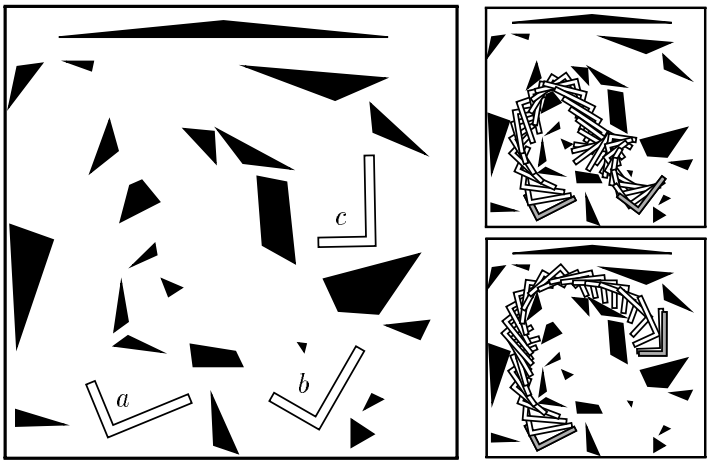
The (maximal) number of these walks (per query) and their (maximal) lengths are parameters of the planner, which we denote by, respectively, N_W and L_W.

Node adding heuristics For both the free-flying robots as the articulated robots, we utilize the (run-time) structure of G to identify "difficult" areas in which more "random" nodes are to be added than in others. We increase the chances for node generation in areas (of C-space) where the graph shows disconnectivities (that is, where there are a number of separate connected components present).

For high dof robots it also proves helpful to identify nodes lying in difficult areas by considering the success/failure ratio of the local method. If this ration is low for a particular node (that is, the local planner fails to connect to the node relatively often), this is an indication that the node lies in some difficult area. In this case, more nodes are added in the (near) neighborhood of the node, in order to locally improve the graph connectivity. We say that the node is *expanded* ([10],[12]).

3.2 Experimental results

We have implemented the method for planar free-flying and articulated robots in the way described above, and we present some experimental results obtained with the resulting planners. The implementations are in C++ and the experiments were performed on a Silicon Graphics Indigo^2 workstation with an R4400 processor running at 150 MHZ. This machine is rated with 96.5 SPECfp92 and 90.4 SPECint92.



	0.25 sec.	0.5sec.	0.75 sec.	1.0 sec.	1.25 sec.
(a,b)	35%	55%	85%	95%	100%
(a,c)	20%	90%	100%	100%	100%
(b,c)	15%	75%	85%	100%	100%

Figure 1: Scene 1. An L-shaped free-flying robot and its test configurations are shown. At the top right, we see two paths computed by the planner and smoothed in 1 second. The table gives the experimental results.

In the test scenes used, the coordinates of all workspace obstacles lie in the unit square. Furthermore, in all scenes we have added an obstacle boundary around the unit square, hence no part of the robot can ever move outside this square.

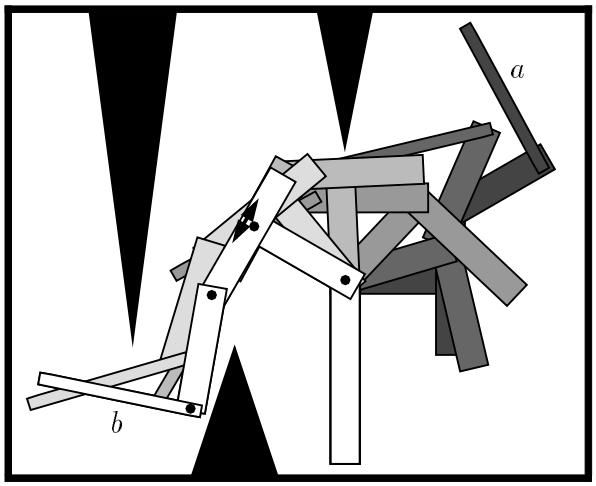
The experiments are aimed at measuring the "knowledge" acquired by the method after having learned for certain periods of time. This is done by testing how well the method solves certain (interesting) queries. For each scene S we define a *query test set* $T_Q = \{(s_1, s_1), (s_2, g_2), \ldots, (s_m, g_m)\}$, consisting of a number of configuration pairs (that is, queries). Then, we repeatedly build a graph by learning for some specified time t, and we count how many of these graphs solve the different queries in T_Q. This experiment is repeated for a number of different learning times t.

The values for the random walk parameters N_W and L_W are, respectively, 10 and 0.05. This guarantees that the time spent per query is bounded by approximately 0.3 seconds (on our machine). Clearly, if we allow more time per query, the method will be more successful in the query phase, and vice versa. Hence there is a trade-off between the learning time and the time allowed for a query.

In Scene 1 (Figure 1) we have a free flying L-shaped robot, placed at the configurations a, b, and c. Experimental results are shown for the three corresponding queries, and two paths are shown, both smoothed in 1 second. We see that around 1 second of learning is required for obtaining roadmaps that solve the queries.

In Scenes 2 to 4 (Figures 2 to 4) results are given for articulated robots.

In the first two scenes, just one query is tested, and well the query (a, b). In both figures,



	2.5 sec.	5sec.	7.5sec.	10 sec.
(a,b)	53.3%	93.3%	100%	100%

Figure 2: Scene 2. A four dof articulated robot, and a path. The table gives the experimental results.

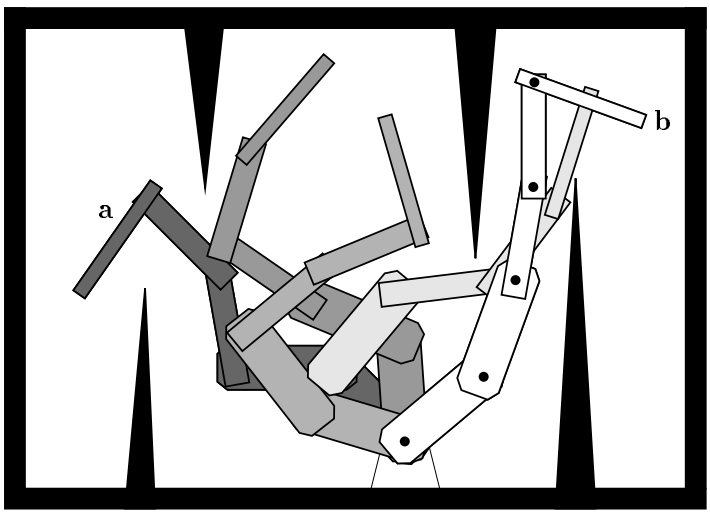
several robot configurations along a path solving the query are displayed using various grey levels. The results of the experiments are given in the two tables. We see that the query in Scene 1 is solved in all 30 cases after having learned for 10 seconds. Learning for 5 seconds however suffices to successfully answer the query in more than 90% of the cases. In Scene 2 we observe something similar.

Scene 4 (Figure 4) is a very difficult one. We have a seven dof robot in a very constrained environment. The configurations a, b, c, and d define 6 different queries, for which the results are shown. These where obtained by a customized implementation by Kavraki and Latombe ([12]). In this implementation, optimized collision checking routines are used, as well as a robot-specific local planner. Furthermore, "difficult" nodes are heuristically identified during the learning phase, and "expanded" subsequently.

4 Application to nonholonomic robots, and experimental results

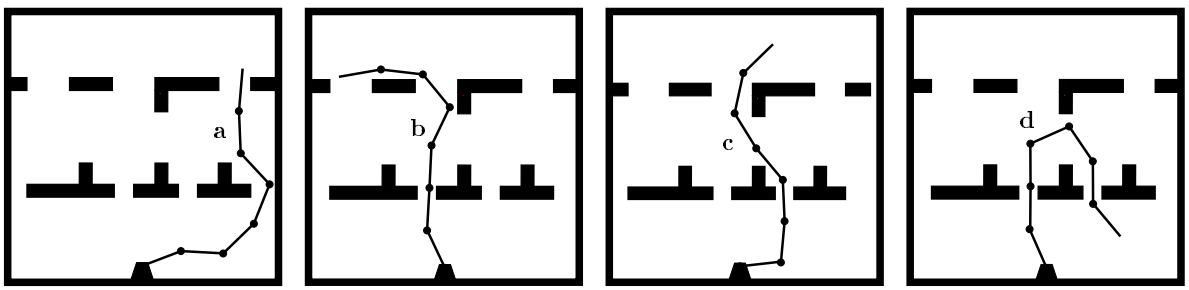
In this section we deal with nonholonomic mobile robots. More specifically, we apply the Probabilistic Path Planner to carlike robots and tractor-trailer robots. Furthermore, we consider two types of carlike robots, i.e., such which can drive both forwards and backwards, and such which can only drive forwards. We refer to the former as *general carlike robots*, and to the latter as *forward carlike robots*. Currently there exist only a few planners for nonholonomic mobile robots which are practical in constrained environments.

In [20] and [17] efficient planners for general carlike robots are described. The approach,



	2.5 sec.	5sec.	7.5sec.	10 sec.
(a,b)	50%	87%	97%	100%

Figure 3: Scene 3. A five dof articulated robot, and a path. Experimental results are shown in the table.



	20 sec.	30 sec.	40 sec.	50 sec.	60 sec.	70 sec.	80 sec.
(a,b)	25%	70%	80%	90%	100%	95%	100%
(a,c)	35%	55%	75%	90%	100%	100%	100%
(a,d)	15%	70%	80%	90%	95%	95%	100%
(b,c)	10%	40%	60%	80%	95%	100%	100%
(b,d)	5%	45%	65%	80%	95%	100%	100%
(c,d)	5%	40%	60%	80%	100%	95%	100%

Figure 4: Scene 4. A seven dof articulated robot in a very constrained environment and the query test set. The table gives the experimental results.

which is applicable to all fully controllable robots[2], consists of three steps. In Step one, given a start configuration s and a goal configuration g, a collision-free path P_1 connecting s and g is computed, without taking into account the non-holonomic constraints. Then, in Step two, a set of configurations $c_1, \ldots, c_n \in P_1$ is picked, such that each c_i can be connected to c_{i+1} by some feasible (simple) path, computed by a local planner. In this way P_1 is transformed into a feasible path P_2, that takes into account the non-holonomic constraints. Finally, in Step three, the resulting path P_2 is smoothed. Step two of this method bares some resemblance with the learning phase of our approach: A global method generates a set of free configurations, and tries to connect certain pairs of these configurations by some local planner. The method exploits the robots full controllable. For non-holonomic robots which are not fully controllable, like for example forward carlike robots, the method cannot be used. Furthermore, the method is not a learning approach. Although some learning method can be used for solving the holonomic problem (step 1), the most time-consuming part of the method is step 2, where holonomic paths are transformed into feasible paths, and such a transformation must be carried out for each query.

Another approach is proposed in [3]. It consists of decomposing the configuration space into an array of small rectangloids and heuristically searching a graph whose nodes are these rectangloids. Two rectangloids are adjacent in this graph if there is a feasible path between a configuration lying in the first rectangloid and a configuration lying in the second rectangloid. The method is suitable for both general carlike robots, as well as forward carlike robots. Furthermore, the method is also applicable to multi-body mobile robots. In practice, however, the complexity of the method becomes overwhelming if the number of trailers exceeds one. Another drawback of the method again is the fact that it is not a learning approach.

We model a carlike robot as a polygon moving in R^2, and its C-space is represented by $R^2 \times [0, 2\pi[$. The motions it can perform are subject to nonholonomic constraints. It can move forwards and backwards, and perform curves of a lower bounded turning radius r_{min}, as an ordinary car. A tractor-trailer robot is modelled as a carlike one, but with an extra polygon attached to it by a revolute joint. Its C-space is (hence) 4-dimensional, and can be represented by $R^2 \times [0, 2\pi] \times [-\alpha_{max}, \alpha_{max}]$, where α is the (symmetric) joint bound. The carlike part (the *tractor*) is exactly a carlike robot. The extra part (the *trailer*) is subject to nonholonomic constraints. Its motions are (physically) dictated by the motions of the tractor (For details, see for example [18]).

For carlike robots, the paths constructed will be sequences of translational paths (describing straight motions) and rotational paths (describing constant non-zero curvature motions) only. It is a well-known fact ([18]) that if for a (general or forward) carlike robot a feasible path in the open free C-space exists between two configurations, then there also exists one which is a (finite) sequence of rotational paths. We include translational paths to enable straight motions of the robot, hence reducing the path lengths. For tractor trailer robots we will use paths that are computed by transformation of the configuration coordinates to the chained form, and using sinusoidal inputs.

4.1 Application to general carlike robots

We now apply the probabilistic learning paradigm, using an undirected graph, to general carlike robots. This asks for filling in some of the (robot specific) details which have been left

[2] A robot is fully controllable iff the existence of a path in the open free C-space is equivalent to the existence of a feasible path.

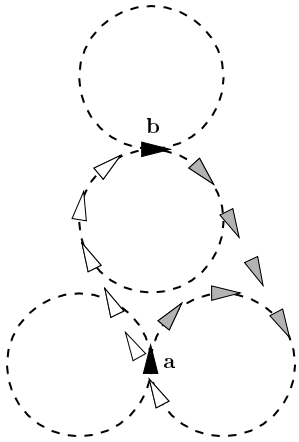


Figure 5: Two RTR paths for a triangular carlike robot, connecting configurations a and b.

open in the discussion of the general method.

4.1.1 Filling in the details

The local planner: A *RTR path* is defined as the concatenation of an rotational path, a translational path, and another rotational path. With this path construct we define the *RTR local planner*: Given two argument configurations a and b, if the shortest RTR path connecting a to b intersects no obstacles, then the planner succeeds and returns this path, and otherwise failure is returned. Figure 5 shows two RTR paths. It can easily be proven that any pair of configurations is connected by a number of RTR paths (See [30] for more details). Furthermore, the RTR local planner satisfies a local topological property which guarantees probabilistic completeness (See Section 5).

An alternative to the RTR local planner is to use a local planner which constructs and tests the shortest (carlike) path connecting its argument configurations ([23], [27]). Constructing shortest carlike paths is however a relatively expensive operation, and the construct requires more expensive intersection checking routines than does the RTR construct. On the other hand, RTR paths will, in general, be somewhat longer than the shortest paths, and, hence, they have a higher chance of intersection with the obstacles. However, the (slightly) higher failure chance of the RTR local planner does not outweigh the outlined advantages.

Collision checking for a RTR path can be done very efficiently, performing three intersection tests for translational and rotational sweep volumes. These sweep volumes are bounded by linear and circular segments (such objects are called *generalized polygons* in [18]) and hence the intersection tests can be done exactly and efficiently. Moreover, the intersection tests for the rotational path segments can be eliminated by storing some extra information in the graph nodes, hence reducing the collision check of a RTR path to just one intersection test for a polygon.

The metric: We use a metric, which is induced by the RTR local planner, and can be regarded as an approximation of the (too expensive) induced "sweep volume metric",

as described in Section 2.1. The distance between two configurations is defined as the length (in workspace) of the shortest RTR path connecting them. We refer to this metric as the *RTR metric*, and we denote it by D_{RTR}.

The random walks in the query phase: Random walks, respecting the carlike constraints, are required. The (maximal) number of these walks (per query) and their (maximal) lengths are parameters of the method, which we denote by, respectively, N_W and L_W.

Let c_s be the start configuration of a random walk. As mentioned above, the parameter L_W defines the maximal length of the walk. As actual length l_W of the walk we take a random value from $[0, L_W]$. The random walk is now performed in the following way: First, the robot is placed at configuration c_s, and a random steering angle ψ and a random velocity v are chosen. Then, the motion defined by (ψ, v) is performed until either a collision of the robot with an obstacle occurs, or the total length of the random walk has reached l_W. In the former case, a new random control is picked, and the process is repeated. In the latter case, the random walk ends.

Good values for N_W and L_W must be experimentally derived (the values we use are given in the next section).

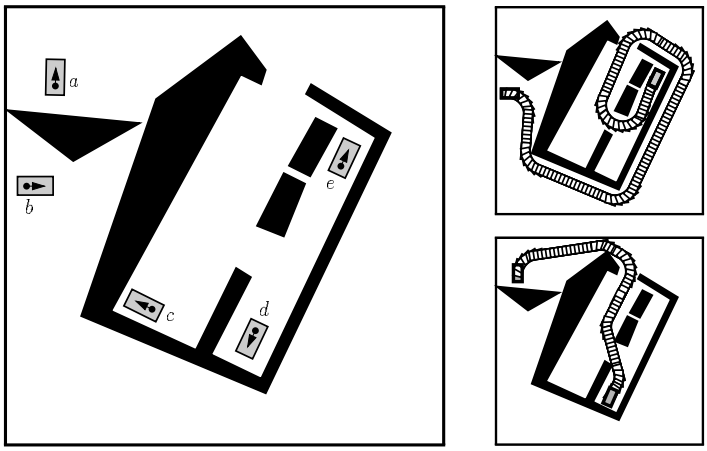
Node adding heuristics We use the geometry of the workspace obstacles to identify areas in which is advantageous to add some extra, geometrically derived, non-random nodes. Particular obstacle edges and (convex) obstacle corners define such geometric nodes (See [31] for more details). Furthermore, as for free-flying robots, we use the the (run-time) structure of the graph G in order to guide the node generation.

4.1.2 Experimental results

We have implemented the planner as described above, and some experimental results are presented in this section. The planner was run on a machine as described in Section 3. Again the presented scenes correspond to the unit square with an obstacle boundary, and the chosen values for N_W and L_W are, respectively, 10 and 0.05. The experimental results are presented in the same form as for the holonomic robots (in Section 3). That is, for different learning times we count how often graphs are obtained which solve particular, predefined, queries.

Scene 5 is a relatively easy scene. It is shown, together with the robot $\mathcal{A}$ positioned at a set of configurations $\{a, b, c, d, e\}$, in Figure 6. The topology is simple and there are only a few narrow passages. As query test set T_Q we use $\{(a, b), (a, d), (b, e), (c, e), (d, e)\}$. (At the top-right of Figure 6 paths solving the queries (a, d) and (b, e), smoothed in 1 second, are shown.) The minimal turning radius r_{min} used in the experiments is 0.1, and the neighborhood size *maxdist* is 0.5. We see that after only 0.3 seconds of learning, the constructed networks solve each of the queries in most cases (but not all). Half a second of learning is sufficient for solving each of the queries, in all 20 trials.

Scene 6, which is shown in Figure 7 (again together with a robot $\mathcal{A}$ placed at different configurations $\{a, b, c, d\}$) is a completely different type of scene. It contains many (small) obstacles and is not at all "corridor-like". Although many individual motion planning problems in this scene are quite simple, the topology of the free C-space is quite complicated, and can only be captured well in relatively complicated graphs. As query test set T_Q we use $\{(a, b), (a, c), (a, d), (c, d)\}$. Furthermore, as in the previous scene, $r_{min} = 0.1$ and *maxdist* $= 0.5$. Again, we show two (smoothed) paths computed by our planner (solving



	0.1 sec.	0.2 sec.	0.3 sec.	0.4 sec.	0.5 sec.
(a,b)	20%	90%	100%	100%	100%
(a,d)	35%	55%	85%	95%	100%
(b,e)	15%	75%	85%	100%	100%
(c,e)	60%	90%	100%	100%	100%
(d,e)	50%	85%	95%	100%	100%

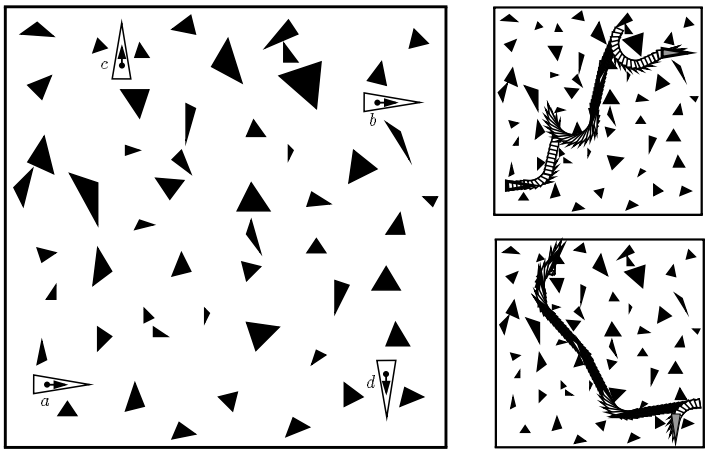
Figure 6: Scene 5, and its test configurations. At the top right, two paths computed by the planner and smoothed in 1 second are shown. The table gives the experimental results.

the queries (a, b) and (c, d)). We see that about 2 seconds of learning are required to obtain networks which are (almost) guaranteed to solve each of the queries.

4.2 Application to forward carlike robots

Forward carlike robots, as pointed out before, lack the reversibility property. Hence, as explained in Section 2.3, directed instead of undirected graphs are used for storing the roadmaps. For details regarding the exact definition of the learning algorithm we refer to [22].

The robot specific components, such as the local planner, the metric, and the random walks are quite analog to those for general carlike robots, as described in Section 4.1. The local planner constructs the shortest *forward RTR path* connecting its argument configurations. A forward RTR paths is defined exactly as a normal RTR path, except that the rotational and translational paths are required to describe *forward* robot motions. The distance between two configurations is defined as the (workspace) length of the shortest forward RTR path connecting them. A random walk is performed as for general carlike robots, with the difference that the randomly picked velocity must be positive, and that, when collision occurs, the random walk is resumed from a random configuration on the previously followed trajectory (instead of from the configuration where collision occurred).



	0.25 sec.	0.5 sec.	0.75 sec.	1.0 sec.	1.5 sec.	2.0 sec.
(a, b)	30%	75%	95%	95%	100%	100%
(a, c)	20%	70%	85%	95%	100%	100%
(a, d)	50%	85%	95%	100%	100%	100%
(c, d)	35%	55%	75%	90%	90%	100%

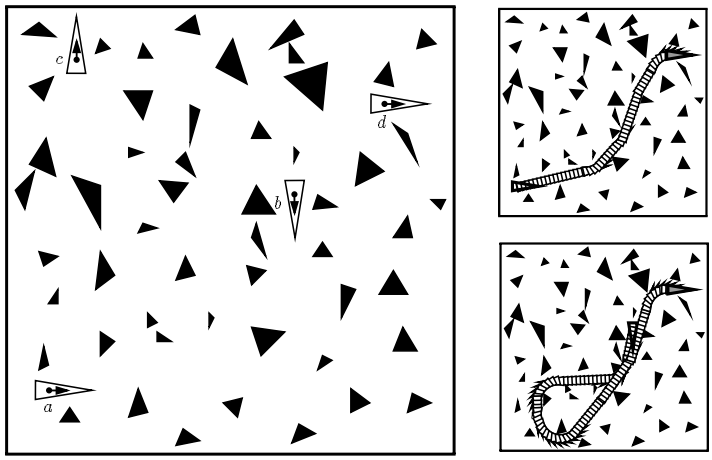
Figure 7: Scene 6, and its test configurations. At the top right, two paths computed by the planner and smoothed in 1 second are shown. The table gives the experimental results.

4.2.1 Experimental results

We give results for Scene 6, of the form used throughout this chapter. We see in Figure 8 that the queries are most likely to be solved after 5 seconds of learning, and (almost) surely after 7.5 seconds. This means that about four times more learning time is required than for general carlike robots.

4.3 Application to tractor-trailer robots

As last example of nonholonomic robots, we now (briefly) consider tractor-trailer robots, and well such which can drive both forwards and backwards. These robots have symmetrical control systems and, hence, undirected underlying graphs are sufficient. Again, we will not go into many details (we refer to [26] for a more thorough discussion of the topic). We use a local planner, by Sekhavat and Laumond ([25]), which transforms its configuration coordinates into the chained form, and uses sinusoidal inputs. We refer to it as the *sinusoidal local planner*. This local planner verifies a local topological property which guarantees probabilistic completeness of the global planner. As distance measure we use a (cheap) approximations of the workspace lengths of the local paths. The random walks in the query phase are basically as those for general carlike robots, except that the trailers orientation must be kept track of during each (constant curvature) motion of the tractor. If, during such a motion, the tractors orientation gets out of bounds (relative to the orientation of the tractor), this is treated as a collision.



	1.0 sec	2.0 sec	3.0 sec	4.0 sec	5.0 sec	7.5 sec
(a, c)	35%	40%	75%	90%	90%	100%
(a, d)	30%	70%	85%	100%	100%	100%
(b, c)	15%	35%	55%	80%	95%	100%
(b, d)	20%	45%	45%	70%	90%	100%

Figure 8: Scene 6 with a forward carlike robot.

4.3.1 Experimental results

See Figure 9 for two feasible paths computed by the Probabilistic Path Planner. The computation time of the roadmap from which the paths where retrieved took about 10 seconds (on the average).

5 On probabilistic completeness

In this section we consider some aspects regarding probabilistic completeness of planners obtained by application of the probabilistic paradigm. A path planner is called *probabilistically complete* iff any problem which is solvable in the open free C-space (that is, without any robot-obstacle contacts) will be solved by the planner provided that it is executed for a sufficient amount of time. For ease of presentation we introduce some shorthand notations. We refer here to the probabilistic path planner by *PPP*. We specify the version using undirected underlying graphs (respectively directed graphs) by PPP_u (respectively PPP_d). The notation $PPP_u(L)$ (respectively $PPP_d(L)$) is used for referring to PPP_u (respectively PPP_d) with a specific local planner L. Throughout this section, we assume that the local planner L is simply a function that takes two argument configurations, and returns a path connecting them which is feasible in absence of obstacles. We say L is *symmetric* iff, for arbitrary configurations a and b, $L(a,b)$ equals $L(b,a)$. So no collision checking is incorporated in the local planner itself[3]. This simplifies the presentation in this section.

We point out that with *PPP* one obtains a probabilistically complete planner for any

[3]Formally this requires a minor adaption of the general outline of the learning algorithm, as presented in Section 2.1. Line (7) will be: **if** $\neg$*connected*$(c,n) \wedge L(c,n) \subset$ *free C-space* **then** $E = E \cup \{(c,n)\}$

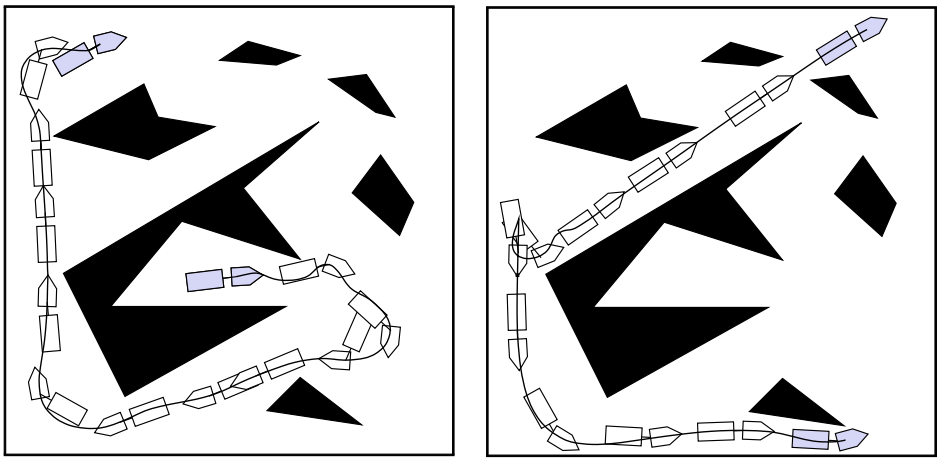


Figure 9: Two feasible paths for a tractor trailer robot, obtained in 10 seconds.

robot which is *small-time locally controllable* (see below), provided that one defines the local planner properly. If, in addition to the small-time local controllability, the robot also has a symmetric control system then $PPP_u(L)$ is suitable, otherwise $PPP_d(L)$ must be used. In Section 5.1 we define a general property for local planners that is sufficient for probabilistic completeness of PPP, and we point out that, given the small-time local controllability of the robot, a local method satisfying this property always exists (but it must be found). We also mention a relaxation of the property, that guarantees probabilistic completeness of $PPP_u(L)$ as well, for small-time local controllable robots with symmetric control systems. All holonomic robots, as well as for example general carlike robots and tractor-trailer robots, fall into this class. Forward carlike robots however are not small-time locally controllable (and neither symmetric). In Section 5.2 we show that all the planners described in this chapter are probabilistically complete.

First however we define the concept small-time local controllability, adopting the terminology introduced by Sussman([28]). Given a robot $\mathcal{A}$, let $\Sigma_{\mathcal{A}}$ be its control system. That is, $\Sigma_{\mathcal{A}}$ describes the velocities that $\mathcal{A}$ can attain in C-space. For a configuration c of a robot $\mathcal{A}$, the set of configurations that $\mathcal{A}$ can reach within time T is denoted by $A_{\Sigma_{\mathcal{A}}}(\leq T, c)$. $\mathcal{A}$ is defined to be *small-time locally controllable* iff for any configuration $c \in \mathcal{C}$ $A_{\Sigma_{\mathcal{A}}}(\leq T, c)$ contains a neighborhood of c (or, equivalently, a ball centered at c) for all $T > 0$. It is a well-known fact that, given a configuration c, the area a small-time locally controllable robot $\mathcal{A}$ can reach without leaving the ϵ-ball around c (for any $\epsilon > 0$) is the open ϵ-ball around c.

5.1 The general local topology property

We assume now that robot $\mathcal{A}$ is small-time locally controllable. For probabilistic completeness of PPP a local planner L is required that exploits the small-time local controllability of $\mathcal{A}$. This is the case if L has what we call the *general local topological property*, or *GLT-property*, as defined in Definition 3 using the notion of *ϵ-reachability* introduced in Definition 2. We denote the ball (in C-space) of radius ϵ centered at configuration c by $B_\epsilon(c)$, and we denote the set of all such balls by $\mathcal{B}_\epsilon$.

Definition 2 *Let L be a local planner for $\mathcal{A}$. Furthermore let $\epsilon > 0$ and $c \in \mathcal{C}$ be given. The* ϵ-reachable area of c by L, *denoted by $R_{L,\epsilon}(c)$, is defined by*

$$R_{L,\epsilon}(c) = \{\tilde{c} \in B_\epsilon(c) | L(c, \tilde{c}) \text{ is entirely contained in } B_\epsilon(c)\}$$

Definition 3 *Let L be a local planner for $\mathcal{A}$. We say L has the GLT-property iff*

$$\forall \epsilon > 0 : \exists \delta > 0 : \forall c \in \mathcal{C} : B_\delta(c) \subset R_{L,\epsilon}(c)$$

We refer to $\mathcal{B}_\delta(c)$ as the ϵ-reachable δ-ball *of c.*

A local planner verifying the GLT-property, at least in theory, always exists, due to the robots small-time local controllability. Theorem 1 now states that this property is sufficient to guarantee probabilistic completeness of *PPP*. That is, of $PPP_u(L)$ if L is symmetric, and of $PPP_d(L)$ otherwise.

Theorem 1 *If L is a local planner verifying the GLT-property, then* PPP(L) *is probabilistically complete.*

Proof
The theorem can be proven directly quite straightforwardly (for both $PPP_u(L)$ and $PPP_d(L)$). Assume L verifies the GLT-property. Given two configurations s and g, lying in the same connected component of the open free C-space, take a path P which connects s and g and lies in the open free C-space as well. Let ϵ be the C-space clearance of P (that is, the minimal distance between P and a C-space obstacle), and take $\delta > 0$ such that $\forall c \in \mathcal{C} : B_\delta(c) \subset R_{L,\epsilon}(c)$. Then, consider a covering of P by balls of radius $\frac{1}{2}\delta$. If every such ball contains a node of V (this is guaranteed to be the case within a finite amount of time), it follows that G contains a path connecting s and g (we assume here that $\{s, g\} \subset V$). □

Clearly, given a small-time locally controllable robot, the GLT-property is a proper criterion for choosing the local planner (sufficient conditions for small-time local controllability of a robot are given in, e.g., [29]). Path planning among obstacles for carlike robots using local planners with the GLT-property has also been studied by Laumond ([19],[9]).

For small-time locally controllable robots with symmetric control systems, a weaker property exists that guarantees probabilistic completeness as well. We refer to this property as the *LTP-property*. The basic relaxation is that we no longer require the ϵ-reachable δ-ball of a configuration a to be centered around c. We do however make a certain requirement regarding the relationship between configurations and the corresponding ϵ-reachable δ-balls. Namely, it must be described by a *Lipschitz continuous function*. For a formal definition of the LTP-property and a proof of probabilistic completeness with local methods verifying it, we refer to [31].

5.2 Probabilistic completeness with the used local planners

The local planners used for holonomic robots, general carlike robots and tractor-trailer robots, as described in this chapter, guarantee probabilistic completeness.

The used local planner L for the holonomic planners constructs the straight line path (in C-space) connecting its argument configurations. It immediately follows that $R_{\epsilon,L}(c) = B_\epsilon(c)$, for any configuration c and any $\epsilon > 0$. Hence, clearly L verifies the GLT-property.

Theorem 2 $\mathrm{PPP}_u(L)$, *with L being the general holonomic local planner, is probabilistically complete for all holonomic robots.*

The planner for general carlike robots uses the RTR local planner. One can prove that this planner verifies the LTP-property ([31]). Again, as stated in the following theorem, this guarantees probabilistic completeness.

Theorem 3 $\mathrm{PPP}_u(L)$, *with L being the RTR local planner, is probabilistically complete for general carlike robots.*

As pointed out before, the theory of the previous sections applies only to robots which are small-time locally controllable. If a robot does not have this property, a local method verifying the GLT-property will in general not exist. A local planner verifying the weaker LTP-property may exist, but this planner will not be symmetric (this would imply the existence of a local planner verifying GTP).

Forward carlike robots are not small-time locally controllable. One can nevertheless prove probabilistic completeness of $PPP_d(L)$, with L being the RTR forward local planner. That is, one can prove that, given two configurations s and g such that there exists a feasible path in the open free C-space connecting them, $PPP_d(L)$ will surely solve the problem within finite time. The proof however does not directly generalize to other cases.

Theorem 4 $\mathrm{PPP}_d(L)$, *with L being the RTR forward local planner, is probabilistically complete for forward carlike robots.*

We give only a sketch of the proof here. Let L be the RTR forward local planner. Assume P_1 is a path in the open free C-space connecting a (start) configuration s to a (goal) configuration g, which is feasible for our forward carlike robot $\mathcal{A}$. Then, one can prove, there exists also a feasible path P_2 in the open free C-space, connecting s to g, which consists of (a finite number of) straight line segments and circular arcs, such that no two distinct arcs are adjacent [4]. In other words, P_2 is of the form $Q_1Q_2Q_3\ldots Q_m$, where Q_i is straight segment if i is even, and an arc otherwise. Let $\{c_1,\ldots,c_{m-1}\}$ be the configurations corresponding to the joining points of the arcs and straight segments, i.e., c_i joins Q_i with Q_{i+1}. Furthermore, let $c_0 = s$ and $c_m = g$. Take $\epsilon > 0$ such that no obstacle, in C-space, lies closer than ϵ to P_2. One can now prove that there exists a $\delta > 0$ such that each $B_\epsilon(c_i)$ (with $i \in \{0,\ldots,m\}$) contains a δ-ball B_i, such that:

$$\forall i \in \{1,\ldots,m-2\} : \forall (a,b) \in B_i \times B_{i+1} : L(a,b) \text{ lies within distance } \epsilon \text{ of } Q_i \ ^5$$

It follows that when a node of G is present in every δ-ball $\subset \mathcal{C}_f$, G will contain a path connecting s to g. We know, due to the probabilistic nature of the node adding, the probability of obtaining such a graph grows to 1 when the learning time goes to infinity.

Regarding tractor-trailer robots, Sekhavat and Laumond prove in [25] that the sinusoidal local planner, used for the tractor-trailer robots, verifies the GLT-property. Hence, for tractor-trailer robots we also have probabilistic completeness.

[4] This does not necessarily hold if P_1 consists of just one or two circular arcs of maximal curvature. In this case however P_1 can be found directly with the local planner.

[5] We say a path Q lies within distance ϵ of a path R, iff $\forall q \in Q : \exists r \in R : |q - r| \leq \epsilon$ (in C-space)

6 A multi-robot extension, and experimental results

A challenging problem in robotics is the multi-robot path planning problem. A number of robots are to change their positions through feasible motions in the same static environment, while avoiding (mutual) collisions. We assume that the robots are identical, although the presented technique is conceptually applicable to problems involving non-identical robots as well.

The multi-robot path planning problem has received a considerable amount of attention in the recent years ([4], [2], [8], [24]). Current approaches basically fall into two classes: *centralized planning methods* and *decoupled planning methods* (See also ([18], [7]). The former are very straight-forward. The idea is that one treats the separate (simple) robots as one composite robot, hence transforming the multi-robot problem into a single-robot one (with many degrees of freedom). Standard motion planning methods can then be used for finding a path in the configuration space of the composite robot. A major drawback however is that the dimension of this configuration space is usually rather large, and, as a result, the time complexity of centralized planning methods is high. Decoupled planning methods plan the paths for the individual robots more or less independently, and, in a second stage, coordinate these paths in a way that mutual robot collisions are avoided. This scheme significantly reduces the amount of computation, but completeness is lost. For example, when two robots are to swap their positions, then they typically follow the same route. Obviously, any coordination of the robot motions along the route will result in collisions.

An extension of the probabilistic paradigm, as presented in this chapter, for solving multi-robot path planning problems is described in this section. It does not fall into either of the two above mentioned classes of multi-robot planning methods. The notion of composite robots is used, but, unlike current centralized approaches, no computations are performed in the configuration space of the composite robot. A roadmap for the composite robot is extracted from information stored in a simple roadmap, computed by the single-robot method for the underlying simple robot. This gives a very flexible scheme, in the sense that it is easily applicable to many different robot types. Furthermore, the resulting planners are probabilistically complete (provided that the local planner for the simple robot is defined properly, as described in the previous section). In this section an application to carlike robots is described, and some experimental results are given.

6.1 Formalization and discretization of the multi-robot planning problem

First we formalize the multi-robot path planning problem. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be n instances of some robot $\mathcal{A}$, present in a workspace $\mathcal{W}$, together with a set of obstacles whose union we denote by $\mathcal{B}$. Furthermore, let $\mathcal{C}$ be the space of all possible configurations of $\mathcal{A}$, and let $\mathcal{C}_f$ be a the subset of $\mathcal{C}$ consisting of all configurations c such that $\mathcal{A}$ placed at c intersects no obstacles. That is, $\mathcal{C}_f$ is $\mathcal{A}$'s free C-space. Given a configuration c, we denote the workspace-area occupied by $\mathcal{A}$, when placed at c, by $\mathcal{A}(c)$.

Definition 4 *A path planning problem for $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is defined as follows: Given start configurations $s_1, \ldots, s_n$ and goal configurations $g_1, \ldots, g_n$ (with $s_i, g_i \in \mathcal{C}_f$), find continuous maps $P_1, \ldots, P_n \in [0,1] \rightarrow \mathcal{C}_f$ describing feasible motions for $\mathcal{A}$, such that ($\forall i, j \in \{1, \ldots, n\}$):*

- $P_i(0) = s_i \wedge P_i(1) = g_i$
- $\forall_{t \in [0,1]} : \mathcal{A}(P_i(t)) \cap \mathcal{B} = \emptyset$

- $\forall_{t\in[0,1]} : i \neq j \Rightarrow \mathcal{A}(P_i(t)) \cap \mathcal{A}(P_j(t)) = \emptyset$

We refer to such a problem simply as problem $((s_1, \ldots, s_n), (g_1, \ldots, g_n))$.

The concept we use is that first a simple roadmap G is computed for $\mathcal{A}$ using the single robot method, and subsequently a roadmap for the composite robot is extracted from G. The used single robot method differs in one point from the one described in Section 2, namely that edges which create cycles are added also. We try to connect to all nodes within distance *maxdist*. This is necessary for the probabilistic completeness of the multi-robot method.

Given a graph $G = (V, E)$ storing a simple roadmap for robot $\mathcal{A}$ (computed by the probabilistic single-robot method), we are interested in solving multi-robot problems using G. We assume that the used local planner for $\mathcal{A}$ guarantees probabilistic completness of the single-robot method. Furthermore, for ease of presentation, we assume that all start configurations s_i and goal configurations g_i are nodes of G (they can always be added as extra, non-random, nodes), and that, for each node $c \in V$ the edge (c, c) is contained in E. We denote the workspace-area swept by $\mathcal{A}$ when moving along a path corresponding to an edge $e \in E$ by $\mathcal{A}(e)$, and we refer to it as *e's sweep-volume*. The idea is that we seek paths in G along which the robots can go from their start configurations to their goal configurations, but we disallow simultaneous motions along paths corresponding to edges e_1 and e_2 with intersecting sweep-volumes. In this way, we avoid mutual robot collisions, while robot-obstacle collisions are ruled out by the fact that we move along the simple roadmap. We say that we *discretize* the multi-robot planning problem to G. More formally, we define a G-discretized path for the composite robot to be a sequence of robot motions such that, at any time instant, at most one robot moves along a local path of G, while all the others are stationary at nodes of G.

We first state that solving G-discretized path planning problems (instead of continuous ones) is sufficient, in the sense that this guarantees probabilistic completeness. Given a set of free configurations W and a graph G computed by the single-robot method, we denote by $G \oplus W$ the graph that is obtained by adding the elements of W to G, as is done with the random configurations.

Theorem 5 *Let* $((s_1, \ldots, s_n), (g_1, \ldots, g_n))$ *be an arbitrary problem for the composite robot, for which there exists a solution in the open free C-space of the composite robot (That is, one without robot-robot and robot-obstacle contacts). Then, within a finite amount of time, the probabilistic single-robot method will produce a graph* G *such that a* $\tilde{G}$*-discretized path solving the problem exists, where* $\tilde{G} = G \oplus \{s_1, \ldots, s_n, g_1, \ldots, g_n\}$.

Theorem 5 states that, given an arbitrary solvable problem for the composite robot, the probabilistic single-robot method will, within a finite amount of time, construct a graph G with which the problem can be solved, provided that we have means for finding G-discretized paths.

6.2 The multi-robot method

The question now is, given a simple roadmap $G = (V, E)$ for a robot $\mathcal{A}$, how to compute G-discretized paths for the composite robot $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ (with all $\mathcal{A}_i$ identical to $\mathcal{A}$). For this we introduce the notion of *super-graphs*. We say an edge $e \in E$ is *blocked* by a node $x \in V$ if $\mathcal{A}(e) \cap \mathcal{A}(x) \neq \emptyset$.

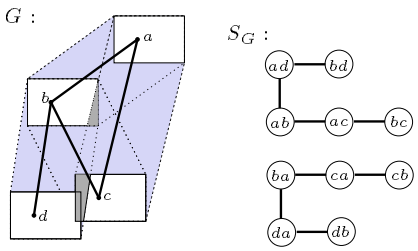


Figure 10: At the left we see a simple roadmap G for the shown rectangular robot $\mathcal{A}$ (shown in white, placed at the graph nodes). We assume here that $\mathcal{A}$ is a translational robot, and the areas swept by the local paths corresponding to the edges of G are indicated in light grey. At the right, we see the G-induced supergraph S_G for $n = 2$. It consists of two separate connected components.

Definition 5 *Given a simple roadmap $G = (V, E)$, the* G-induced supergraph $S_G = (V_S, E_S)$ *is defined as follows:*

- $(x_1, \ldots, x_n) \in V^n$ *is a node of S_G iff*
  $i \neq j \Rightarrow \mathcal{A}(x_i) \cap \mathcal{A}(x_j) = \emptyset$. *We refer to the nodes of S_G as* super-nodes. *Given a super-node $X = (x_1, \ldots, x_n)$, we refer to the x_i's as X's* underlying simple nodes.
- *Two super-nodes $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ are connected by an edge of S_G if and only if for exactly one $i \in \{1, \ldots, n\}$ $x_i \neq y_i$, and x_i is connected to y_i by an edge in G which is not blocked by an x_j, with $j \neq i$. We refer to the edges of S_G as* super-edges.

So each node of S_G corresponds to a feasible placement of the n simple robots at nodes of G, and each edge of S_G corresponds to a feasible motion of one simple robots along an edge of G. See Figure 10 for an example of a (simple) G-induced supergraph. Any path in the G-induced supergraph describes a simple G-discretized path (for the composite robot), and vice-versa. Hence, the problem of finding G-discretized paths for our composite robot reduces to graph searches in S_G.

The size of a G-induced super-graph, as defined above, is exponential in n (the number of robots). However, the entire data-structure does not have to be stored explicitly. Given a particular super-node X, its neighbors in S_G can be retrieved in constant time provided that, for each $(x, e) \in V \times E$, we know whether $\mathcal{A}(x)$ intersects $\mathcal{A}(e)$. This asks for a data-structure of quadratic size (in the size of G) which for each node-edge pair (x, e) stores whether $\mathcal{A}(x) \cap \mathcal{A}(e) = \emptyset$. Using optimized intersection routines, such a data-structure, which we refer to as the *G-intersection map*, can be computed and updated quite efficiently. Hence, for performing graph searches in S_G, we need only to compute and store the set V_S of super-nodes. If however n is large, then the required amount of memory can still be very (too) large. Such cases ask for reducing the number of super-nodes. For techniques achieving this we refer to [32].

Our multi-robot approach for solving a composite problem $((s_1, \ldots, s_n), (g_1, \ldots, g_n))$ now consists of the following steps:

1. Compute a simple roadmap G of sufficient density using the probabilistic single robot approach (allow for cycles).
2. Add $s_1, \ldots, s_n$ and $g_1, \ldots, g_n$ to G (together with edges connecting them to other nodes).
3. For each node v and each edge e, compute and store whether $\mathcal{A}(v) \cap \mathcal{A}(e) = \emptyset$ (That is, compute the G-intersection map).
4. Construct the supergraph S_G as described above, and store it in a partially implicit form.
5. Find the shortest path in S_G between node $(s_1, \ldots, s_n)$ and node $(g_1, \ldots, g_n)$.
6. Transform this path into a feasible path in the configuration space of the composite robot, using the local planner for $\mathcal{A}$.
7. Smooth the (maximal) segments of the composite path where only one robot moves.
8. Combine consecutive motions of different simple robots into simultaneous ones, as far as this is possible.

The last two steps of the algorithm require some explanation. In step (7) we identify maximal segments of the (composite) path where just one (simple) robot moves. Each such segment can be smoothed with the use of standard single-robot smoothing techniques, after the stationary robots have (temporarily) been added to the set of obstacles. Typically, this technique significantly reduces the length of the composite path. However, it does not allow for simultaneous motions of the simple robots. In Step (8) we heuristically identify segments in the composite path where the consecutive robot-motions can be replaced by simultaneous ones, without introducing mutual robot collisions. This step again reduces the length of the composite path.

6.3 Application to multiple carlike robots and experimental results

The described method is very general. The only robot-specific components are, as in the single robot method, the local planner and a (induced) metric. We have implemented a multi-robot motion planner for carlike robots, based on the super-graph concept as described in the previous section. In this implementation, we have interleaved the construction of the simple roadmap and that of the super-graph (Steps 1,3, and 4). That is, whenever a simple node is added to G, we immediately extend the super-graph S_G correspondingly. In this way we obtain a method that is truly probabilistically complete; if it runs long enough then any problem which is solvable (in the open free C-space) will be solved. The simple roadmaps are computed using the method as described in Section 2, with, as stated before, the difference that we also add edges which create cycles. So, we try to connect to all nodes within distance *maxdist*, with respect to the RTR metric.

We have done experiments on a Silicon Graphics Indigo^2 workstation rated with 96.5 SPECfp92 and 90.4 SPECint92. We present some results for 2 scenes.

See Figure 11 for Scene 1. It consists of a narrow "H-shaped" corridor, and there are three rectangular carlike robots present. The problem in the scene is that, if one robot is to change its position, the others often have to make large detours. A decoupled planning method will not solve such problems.

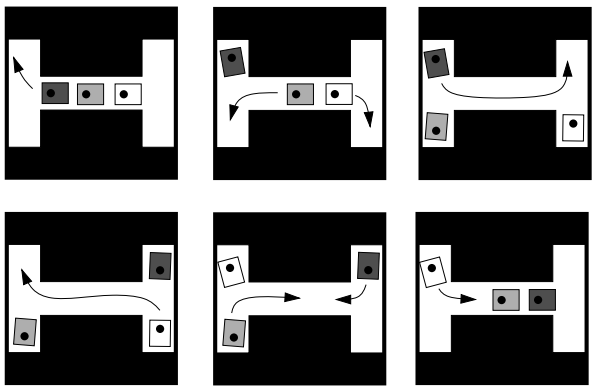


Figure 11: Scene 1. Three carlike robots in a narrow corridor. The white robot has to swap places with the dark robot.

Within 2 seconds a super-graph was obtained which solved most problems in this scene, among which the shown "swapping-problem" (Figure 11). This super-graph contained about 20.000 super-nodes. The computation of a shortest path in S_G took about 1 second, as did the smoothing of the path.

Scene 2 (Figure 12), which also involves 3 carlike robots, required larger super-graphs to be constructed. This is due to the fact that, in comparison with the previous scene, the free space is quite large, and, hence, a larger number of simple nodes (i.e., nodes of the simple roadmap) are required to obtain a sufficient "covering". The problem shown required about 4 seconds for the construction of a suitable super-graph (which contained about 150.000 nodes), and another 5 seconds for the shortest-path search in the super-graph. Again, smoothing took about 1 second.

7 Conclusions

A general probabilistic technique for path planning has been described in this chapter. It consists of two phases. In the learning phase a probabilistic roadmap is incrementally constructed, which can subsequently, in the query phase, be used for solving individual motion planning problems in the given scene. It is a fast and flexible method. In order to apply it to some particular robot type, all one needs to define (and implement) is a local method which computes paths feasible for this robot type, and some (induced) metric. Furthermore, proper choice of the local method guarantees probabilistic completeness. Numerous extensions of the approach are possible. One such extension has been described in this chapter, dealing with the multi-robot path planning problem. Other possibilities include, for example, path planning in partially unknown environments, path planning in dynamic environments (e.g., amidst moving obstacles), and path planning in the presence of movable obstacles.

References

[1] J. M. Ahuactzin, E.-G. Talbi, P. Bessiere, and E. Mazer. Using genetic algorithms for

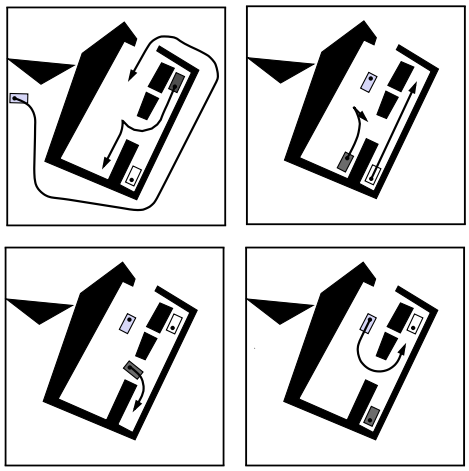


Figure 12: Scene 2. Three carlike robots in a workspace with wide and narrow areas.

robot motion planning. In *10th Europ. Conf. Artific. Intell.*, pages 671–675, London, England, 1992. John Wiley and Sons, Ltd.

[2] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Internat. J. Robot. Res.*, 10:628–649, 1991.

[3] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.

[4] M. Erdmann and T. Lozano-Peréz. On multiple moving objects. Technical Report 883, MIT, Massachusets, USA, 1986.

[5] B. Faverjon and P. Tournassoud. A practical approach to motion planning for manipulators with many degrees of freedom. In *Proc. 5th Intern. Symp. on Robotics Research*, pages 65–73, 1990.

[6] Th. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom - random reflections at c-space obstacles. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, San Diego, USA, 1994.

[7] Y. Hwang and N. Ahuja. Gross motion planning—a survey. *ACM Comput. Surv.*, 24(3):219–291, 1992.

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

[9] P. Jacobs, J.-P. Laumond, and M. Taïx. A complete iterative motion planner for a car-like robot. *Journees Geometrie Algorithmique*, 1990.

[10] L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration space for fast path planning. Technical report, San Diego, USA, 1994.

[11] L. Kavraki, J-C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot motion planning. Technical report, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, December 1995.

[12] L. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *To appear in IEEE Trans. Robot. Autom.*, 1995.

[13] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe. Planning motions with intentions. In *Proc. of SIGGRAPH'94*, 1994.

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

[15] P.L. Kociemba L. Graux, P. Millies and B. Langlois. Integration of a path generation algorithm into off-line programming of airbus panels. Technical report, 1992.

[16] F. Lamiraux and J.P. Laumond. On the expected complexity of random path planning. Technical Report 95087, LAAS, Toulouse, France, March 1995.

[17] J.-C. Latombe. A fast path planner for a car-like indoor mobile robot. In *Proc. 9th Nat. Conf. Artific. Intell.*, pages 659–665, 1991.

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

[19] J.-P. Laumond, M. Taïx, and P. Jacobs. A motion planner for car-like robots based on a mixed global/local approach. In *IEEE IROS*, July 1990.

[20] Jean-Paul Laumond, Paul E. Jacobs, Michel Taïx, and Richard M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Trans. Robot. Autom.*, 10(5), October 1994.

[21] M. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, October 1992.

[22] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In *Proc. The First Workshop on the Algorithmic Foundations of Robotics*, pages 19–37. A. K. Peters, Boston, MA, 1994.

[23] J.A. Reeds and R.A. Shepp. Optimal paths for a car that goes both forward and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[24] J. H. Reif and H. Wang. Social potential fields: A distributed behavioral control for automonous robots. In *Proc. The First Workshop on the Algorithmic Foundations of Robotics*, pages 331–345. A. K. Peters, Boston, MA, 1994.

[25] S. Sekhavat and J.P. Laumond. Topological property of nonholonomic motion planning methods for chained form systems. Technical report, LAAS/CNRS, Toulouse, France, 1995, to appear.

[26] S. Sekhavat, P. Švestka, J.-P. Laumond, and M.H. Overmars. On two-level path planning for tractor-trailer robots using a fictive system. Technical report, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, July 1995, to appear.

[27] P. Souères and J.P. Laumond. Shortest paths synthesis for a car-like robot. Technical report, LAAS/CNRS, Toulouse, France, September 1992.

[28] H.J. Sussman. Lie brackets, real analyticity and geometric control. In R.W. Brockett, R.S. Millman, and H.J. Sussman, editors, *Differential Geometric Control Theory.* Birkhauser, 1983.

[29] H.J. Sussman. A general theorem on local controllability. *SIAM Journal on Control and Optimization*, 25(1):158–194, January 1987.

[30] P. Švestka. A probabilistic approach to motion planning for car-like robots. Technical Report RUU-CS-93-18, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, April 1993.

[31] P. Švestka and M. Overmars. Motion planning for car-like robots using a probabilistic learning approach. Technical Report RUU-CS-94-33, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, May 1994.

[32] P. Švestka and M.H. Overmars. Coordinated motion planning for multiple carlike robots using probabilistic roadmaps. In *Proc. IEEE Internat. Conf. on Robotics and Automation (to appear)*, Nagoya, Japan, 1995.

[33] P. Švestka and J. Vleugels. Exact motion planning for tractor-trailer robots. In *Proc. IEEE Internat. Conf. on Robotics and Automation (to appear)*, Nagoya, Japan, 1995.

[34] Arnaud Vandame. Planification de tâches de manipulations par méthodes de recherches aléatoires. master thesis, LAAS, Toulouse, France, June 1994.