Probabilistic Path Planning

P. Švestka M. H. Overmars

This is the fifth chapter of the book:

Robot Motion Planning and Control
Jean-Paul Laumond (Editor)

Laboratoire d'Analye et d'Architecture des Systèmes
Centre National de la Recherche Scientifique
LAAS report 97438

Previoulsy published as:
Lectures Notes in Control and Information Sciences 229.
Springer, ISBN 3-540-76219-1, 1998, 343p.

Probabilistic Path Planning

P. Švestka and M. H. Overmars

Utrecht University

1 Introduction

The robot path planning problem, which asks for the computation of collision free paths in environments containing obstacles, has received a great deal of attention in the last decades [25,15]. In the basic problem, there is one robot present in a static and known environment, and the task is to compute a collision-free path describing a motion that brings the robot from its current position to some desired goal position. Variations and extensions of this basic problem statement are numerous.

To start with, for a large class of robots (i.e., nonholonomic robots) computation of collision-free paths is not sufficient. Not only are the admissible robot placements constrained by obstacles and the robot geometry, but also are the directions of motion subject to constraints. For example, mobile robots moving on wheels have such nonholonomic constraints, due to the fact that their wheels are not allowed to slide. Another realistic scenario is that of multiple robots acting in the same environment. In this case, apart from the restrictions imposed by the obstacles, robot geometry, and possible nonholonomic constraints, one also has to avoid collisions between the robots mutually. Moving obstacles, uncertainties in sensing, and inexact control add further levels of difficulty.

In order to build robots that can autonomously act in real-life environments, path planning problems as sketched above need to be solved. However, it has been proven that, in general, solving even the basic path planning problem requires time exponential in the robots number of degrees of freedom. In spite of this discouraging problem complexity, various such *complete* planners have been proposed. Their high complexity however makes them impractical for most applications. And every extension of the basic path planning problem adds in computational complexity. For example, if we have n robots of d degrees of freedom each, the complexity becomes exponential in nd. Or if we allow for moving obstacles, the problem becomes exponential in their number [9,35]. Assuming uncertainties in the robots sensing and control, leads to an exponential dependency on the complexity of the obstacles [9].

The above bounds deal with the exact problem, and therefore apply to *complete planners*. These are planners that solve any solvable problem, and return failure for each non-solvable one. So for most practical problems it seems

impossible to use such complete planners. This has lead many researchers to consider simplifications of the problem statement.

A quite recent direction of research, which we just want to mention briefly here, deals with the formulation of assumptions on the robot environment that reduce the path planning complexity. This is based on the belief that there exists a substantial gap between the theoretical worst-case bounds of path planning algorithms and their practical complexity. A number of researchers have attempted to formulate assumptions on the obstacles that prohibit the (artificial) constructions that cause the worst-case bounds. Examples of such assumptions are, amongst others, *fatness* [54,53], *bounded local complexity* [36], and *dispersion* [33]. However, this line of research has been mainly of theoretical nature, and has not yet resulted in implementations of practical path planners. Also, it is currently not clear whether similar results can be obtained for extensions of the basic path planning problem.

Instead of assuming things about the robot environment, many researchers have simply dropped the requirement of completeness for the planner. Heuristic planners have been developed that solve particular difficult problems in impressively low running times. However, the same planners also fail or consume prohibitive time on seemingly simpler ones. For autonomous robots in realistic environments this might be a problem, since one cannot predict the path planning problems such robots will face.

So, on one hand, completeness is a preferred property of motion planners for autonomous robots, while, on the other hand, only heuristic algorithms are capable of solving many of the practical problems that people are interested in. This has lead to the design of path planners that satisfy weaker forms of completeness, in particular *resolution completeness* and *probabilistic completeness*. In this chapter we deal with the latter. A planner is called probabilistically complete if, given a solvable problem, the probability of solving it converges to 1 as the running time goes to infinity. Such a planner is guaranteed to solve any solvable problem within finite time. And if one can estimate the probability of solving a problem with respect to the running time, one has an even stronger result.

Two of the most successful such planners are the *probabilistic path planner* (*PPP*) and the *randomized path planner* (*RPP*).

RPP [5,17] is a potential field planner, that escapes local minima by performing Brownian motions. It has successfully been applied to articulated robots with many degrees of freedom (dof). Also, it has been used for checking whether parts can be removed from aircraft engines for inspection and maintenance, and for automatically synthesising a video clip with graphically simulated human and robot characters entailing a 78-dimensional configuration space. The planner works as follows : Given a goal configuration g, a potential field U is computed, being a function that assigns positive real values to con-

figurations. Roughly, as is the case for potential fields in general, U is defined by an attracting potential of the goal configuration, and repulsing potentials of the obstacles, and can be seen as a landscape with the obstacles as mountains and the goal configuration as lowest point. Connecting the start configuration s to the goal configuration g is attempted by "descending" along U. Such a *down motion* always ends in a local minimum. If this local minimum is g (the global minimum), then the problem is solved. If this is not the case, a Brownian motion is performed, and the process is repeated. Using well-known properties of Brownian motions, *RPP* can be proven to be probabilistically complete [17,24]. Moreover, a calculation of the finite expected number of Brownian motions is given in [24]. This calculation uses the fact that the basins B_i of attraction of the local minima form a partition of the free configuration space. For each pair (B_i, B_j) one can define the transition probability p_{ij} that a Brownian motion, starting at the minimum of B_i, terminates somewhere in B_j. The expected number of Brownian motions can then be expressed as a function of the transition probabilities p_{ij}'s. This nice theoretical result has however the practical drawback that the p_{ij}'s are, in non-trivial cases, unknown. Although *RPP* proves to be very powerful for many practical problems, the method also has some drawbacks. For example, since the planner is potential field based, it does not memories any knowledge about the configuration space after having solved a particular problem, and, for this reason, each new problem requires a whole new search. In other words, it is a single shot approach. Furthermore, it appears to be easy to create seemingly simple problems for which the planner consumes a more than reasonable amount of time, due to very low transition probabilities between certain basins. Also, the method does not apply directly to nonholonomic robots.

Other probabilistically complete planners for static and dynamic environments utilising genetic algorithms are described in [1,7]. Other work on related probabilistic path planning approaches includes [16]. We will not go into details here.

This chapter gives a survey on the probabilistic path planner *PPP*, which is a very general planner, or planning scheme, building probabilistic roadmaps by randomly selecting configurations from the free configuration space and interconnecting certain pairs by simple feasible paths. The method is probabilistically complete and not restricted to any particular class of robots.

A first single-shot version of the planner for free-flying planar robots was described in [31] and subsequently expanded into a general learning approach, for various robot types, in [32]. Independently, "*PPP*-like" preprocessing schemes for holonomic robots where introduced in [21] and [14]. These schemes also build probabilistic roadmaps in the free C-space, but focus on the case of many-dof robots. In [23] the ideas developed in [32] and [21] have been combined, resulting in an even more powerful planner for high-dof robots. Simultaneously, *PPP*

has been applied to nonholonomic robots. Planners for car-like robots that can move both forwards and backwards as well as such that can only move forwards are described in [45,47]. *PPP* applied to tractor-trailer robots is the topic of [49,39]. Probabilistic completeness of the planners for nonholonomic robots is proven in [47]. Recently some first results on the expected running times of *PPP*, under certain geometric assumptions on the free configuration space, have been obtained [22,20,3]. For a thorough survey of probabilistic path planning for holonomic robots we also refer to the thesis of Kavraki [19]. Finally, extensions of *PPP* addressing multi-robot path planning problems have been presented in [46,48].

In this chapter an overview is given of the algorithmic aspects of *PPP* and applications of the planning scheme to various robot types are discussed. Also, some theory is presented regarding probabilistic completeness and expected running times. The chapter is organised as follows: In Section 2 the probabilistic paradigm is described in its general form. In the following two sections the paradigm is applied to specific robot types, i.e., to holonomic robots (free-flying and articulated) in Section 3, and to nonholonomic mobile robots (car-like and tractor-trailer) in Section 4. In both sections the robot specific components of the algorithm are defined, and obtained simulation results are presented. Sections 5 and 6 are of a more theoretical nature. In Section 5 aspects regarding probabilistic completeness of the method are discussed, and proofs of probabilistic completeness are given for the planners described in this chapter. Section 6 deals with analyses of expected running time. Results by Kavraki et al. [22,20,3] are reviewed, and some new results are presented as well. In Section 7 an extension of *PPP* for solving multi-robot path planning problems is described, and simulation results are given for problems involving multiple car-like robots.

2 The Probabilistic Path Planner

The *Probabilistic Path Planner* (*PPP*) can be described in general terms, without focusing on any specific robot type. The idea is that during the *roadmap construction 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 path planning problems.

The data-structure constructed during the roadmap construction 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*. A local planner L is simply a function that takes two configurations as arguments, and returns a path connecting them, that is feasible in absence

of obstacles (that is, the path respects the constraints of the robot). Proper choice of the local planner guarantees probabilistic completeness of the global planner, as we will see in Section 5. If, given two configurations a and b, the path $L(a, b)$ is collision-free, then we will say that L *connects* from a to b.

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. So the paths generated in the query phase (that 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.

2.1 The roadmap construction phase

We assume that we are dealing with a robot $\mathcal{A}$, and that L is a local planner that constructs paths for $\mathcal{A}$. We assume that L is *symmetric*, that is, for any pair of configurations (a, b) $L(a, b)$ equals $L(b, a)$ reversed. See Section 2.3 for remarks on non-symmetric local planners. As mentioned above, in the roadmap construction 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 configuration space (or *free C-space*). We try to connect each generated node c to previously added ones with L, and each such successful connection results in a corresponding edge being added to E.

More precisely, this edge adding is done as follows : First, a set N_c of neighbours is chosen from V. This set consists of nodes lying within a certain distance from c, with respect to some distance measure D. Then, in order of increasing distance from c, we pick nodes from N_c. For each such picked node n, we test whether L connects from c to n, and, if so, (c, n) is added to E. However, if n is already graph-connected with c at the moment that it is picked, n is simply ignored. So 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 roadmap construction 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 that does allow cycles in the graph [32]. These methods however have the disadvantage that they slow down the roadmap construction algorithm, due to the fact that the adding of a node requires more calls of the local planner. Another possibility is to build a forest as described above, but, before using the graph for queries, "smoothing" the graph by adding certain edges that 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 $\mathcal{C}$ denote the C-space of the robot, and $\mathcal{C}_f$ the free portion of $\mathcal{C}$ (i.e., the free C-space). To describe the roadmap construction algorithm formally, we need a function $D \in \mathcal{C} \times \mathcal{C} \rightarrow \mathbf{R}^+$. It defines the distance measure 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 graph $G = (V, E) \in \mathcal{C}_f \times \mathcal{C}_f^2$ is constructed as follows:

The roadmap construction algorithm

(1) $V = \emptyset$, $E = \emptyset$
(2) **loop**
(3) $c =$ a "randomly" chosen free configuration
(4) $V = V \cup \{c\}$
(5) $N_c =$ a set of neighbours of c chosen from V
(6) **forall** $n \in N_c$, in order of increasing $D(c, n)$ **do**
(7) **if** $\neg \mathit{connected}(c, n) \wedge L(c, n) \subset \mathcal{C}_f$ **then** $E = E \cup \{(c, n)\}$

The construction algorithm, as described above, leaves a number of choices to be made: A local planner must be chosen, a distance measure must be defined, and it must be defined what the neighbours 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 roadmap construction phase is the local planner. As mentioned before, the local planner must construct paths that are *feasible* for $\mathcal{A}$, in absence of obstacles. This simply

means that the paths it constructs describe motions that are performable by the robot, that is, motions that respect the robots constraints. For example, assume the robot is a car-like vehicle moving on wheels, then a local planner that just connects the two argument configurations with a straight-line segment (in C-space) is not suitable, since it describes motions that force the wheels of the robot to slide.

Furthermore, we want the roadmap construction algorithm to be fast. For this, it is important that (1) the local planner constructs its paths in a time efficient manner and (2) the probability that local paths intersect with obstacles is low. The first requirement can be met by keeping the path constructs as simple as possible. For obtaining low intersection probabilities, the local planner should construct paths with relatively small *sweep volumes*. That is, the volumes (in workspace) swept by the robot when moving along local paths should preferably be small. Clearly, local planners minimising these sweep volumes also minimise the probabilities of the local paths intersecting obstacles.

Finally, the local planner should guarantee probabilistic completeness of *PPP*. In Section 5 we give sufficient properties.

Neighbours and edge adding methods Another important choice to be made is that of the neighbours 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 roadmap construction algorithm. Reasons for this are that the choice of the neighbours strongly influences the overall structure of the graph, and that, regardless of how the local planner is exactly defined, the calls of the local planner are by far the most time-consuming operations of the roadmap construction algorithm (due to the collision tests that must be performed).

So it is clear that calls 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 that are already in c's connected component are useless. For this reason the roadmap construction algorithm builds a forest. Secondly, local planner calls that 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 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 regarding the actual choice for N_c. We have decided on taking all nodes within distance *maxdist* as neighbours. 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 that are already in c's connected component at the moment that the connection is to be attempted. By considering elements of N_c in this order we expect to maximise 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 neighbours 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 connect 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) of $L(a, b)$, that is, as the volume of the area swept by the robot when moving along $L(a, b)$. In this way each local planner L induces its own distance measure, that 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. However, in the general case, 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 roadmap construction 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 the 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 that 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 car-like robots adding (extra) configurations that correspond to placements of the robot "parallel" to obstacle edges and "around" convex obstacle corners boosts the performance of the roadmap construction algorithm 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 neighbourhood. 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 behaviour of the local planner. For example, if the local planner often fails 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 roadmap construction 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 constructed 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 $\mathit{query\_mapping} \in \mathcal{C} \times \mathcal{C} \rightarrow V \times V$, defined as follows:

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

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

So $\mathit{query\_mapping}(a,b)$ returns the pair of connected graph nodes $(\tilde{a},\tilde{b})$ that minimise 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 $(\tilde{s},\tilde{g}) = \mathit{query\_mapping}(s,g)$, and to try to connect with the local planner from s to $\tilde{s}$ and from $\tilde{g}$ to g. However, since no obstacle avoidance is incorporated in the local planner, it

may, in unlucky cases, fail 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 aim 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 [32]). 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 algorithm outline of *PPP*, as described in the previous section, the computed roadmaps are stored in undirected graphs. For many path 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, path planning problems involving free-flying robots, articulated robots, and (normal) car-like robots can all be dealt with using undirected underlying graphs. There are however path planning problems for which undirected underlying graphs not sufficient, and directed ones are required instead. For example, problems involving car-like robots that can only move forwards require directed underlying graphs.

The existence of an edge (a, b) in the underlying graph G corresponds to the statement that the local planner constructs a feasible path from a to b. If however 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 constructs 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 connects in both directions. Doing so, useful information might be thrown away. This will happen in those cases where the local planner connects in exactly one direction, and the fact that it has successfully constructed a feasible path will not be stored. If however the local planner is *symmetric*, which means that it connects from say a to b whenever it connects from b to 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 that are symmetric, depends on the properties of the robot $\mathcal{A}$, defined by the constraints imposed on it.

Definition 1. *A robot $\mathcal{A}$ is* $\mathcal{C}$-symmetric *(*configuration space symmetric*) if and only if any feasible path for $\mathcal{A}$ remains feasible when reversed.*

All holonomic robots are $\mathcal{C}$-symmetric. For nonholonomic robots this is not the case. For example, a car-like robot that can drive forwards as well as backwards is $\mathcal{C}$-symmetric while one that can only drive forwards is not. In terms of control theory, a (nonholonomic) robot is $\mathcal{C}$-symmetric 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 the velocity $-v$.

Clearly, if $\mathcal{A}$ is $\mathcal{C}$-symmetric, then any local planner L that constructs feasible paths for $\mathcal{A}$ can be made symmetric in a trivial way, by reversing computed paths when necessary. So this implies that for any $\mathcal{C}$-symmetric robot 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 [45] and [47] for discussions on this topic, and sensitive strategies for the adding of directed edges.

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 cycle-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, constructed 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 that builds graphs containing loops. This will result in a slowdown of the roadmap

construction phase, but the gain is that the paths (directly) retrieved in the query phase will be shorter.

3 Application to holonomic robots

In this section an application of *PPP* 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 is either a *prismatic joint*, or a *revolute joint*. If J_i is a prismatic joint, then link L_i can translate along some vector that 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 that 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 $[low_1, up_1] \times [low_2, up_2] \times \cdots \times [low_n, up_n]$. 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 are $\mathcal{C}$-symmetric, 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, discretise 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 a 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 distance measure: 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 roadmap construction phase. Multiple random walks were performed from the query configurations s and g, aimed at connecting the end-points 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 utilise 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 planner. 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) neighbourhood of the node, in order to locally improve the graph connectivity. We say that the node is *expanded* [21],[23].

3.2 Simulation results

We have implemented the method for planar free-flying and articulated robots in the way described above, and we present some simulation 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.

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 move outside this square.

The experiments are aimed at measuring the "knowledge" acquired by the method after having constructed roadmaps 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 construct a graph 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 construction times t. The results are presented in the tables under the figures. The numbers in the boxes indicate the percentage of the runs that solve the corresponding query within the given time bound.

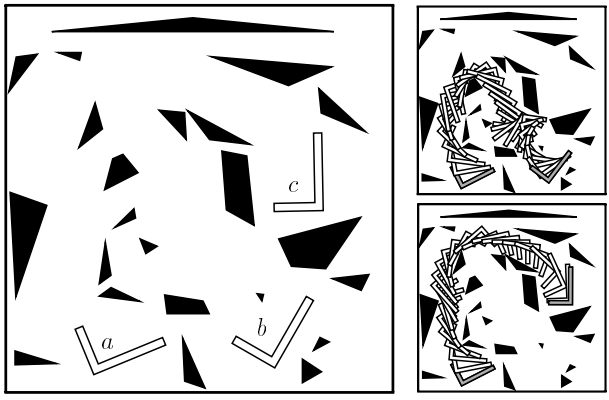
The values for the random walk parameters N_W and L_W are, respectively, 10 and 0.05. This guarantees that the time spent per query is bounded by approximately 0.3 seconds (on our machine). Clearly, if we allow more time per query, the method will be more successful in the query phase, and vice versa.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

In Figure 1 we have a free flying L-shaped robot, placed at the configurations a, b, and c. Simulation 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 roadmap construction is required for obtaining roadmaps that solve the queries. These roadmaps consist of approximately 125 nodes.

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

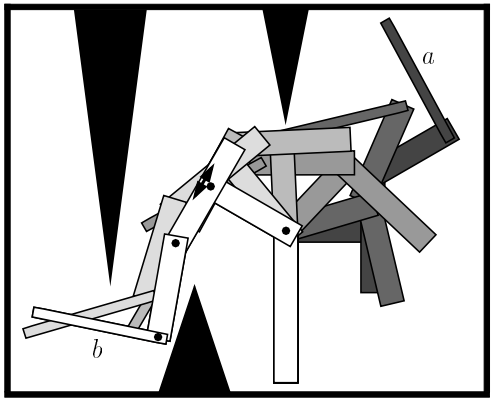
In the first two scenes, just one query is tested, and well the query (a, b). In both figures, several robot configurations along a path solving the query are displayed using various grey levels. The results of the experiments are given in the two tables. We see that the query in Figure 2 is solved in all cases after 10 seconds of construction time. Roadmap construction for 5 seconds however suffices to successfully answer the query in more than 90% of the cases. In Figure 3 we observe something similar. For both scenes the roadmaps constructed in 10 seconds contain around 500 nodes.

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 customised implementation by Kavraki et al. [23]. In this implementation, optimised collision



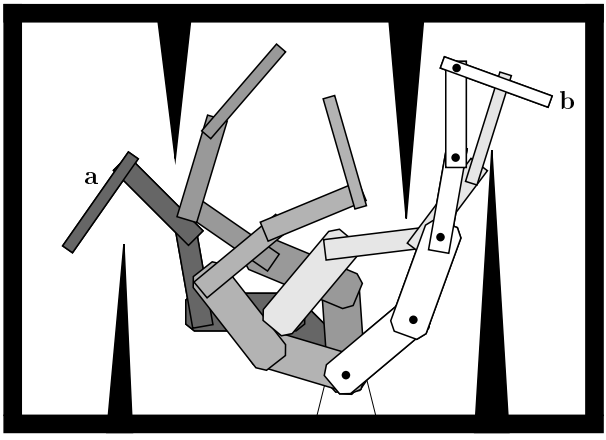
	0.25 sec.	0.5sec.	0.75 sec.	1.0 sec.	1.25 sec.
(a,b)	35%	55%	85%	95%	100%
(a,c)	20%	90%	100%	100%	100%
(b,c)	15%	75%	85%	100%	100%

Fig. 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.



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

Fig. 2. A four dof articulated robot, and a path.



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

Fig. 3. A five dof articulated robot, and a path.

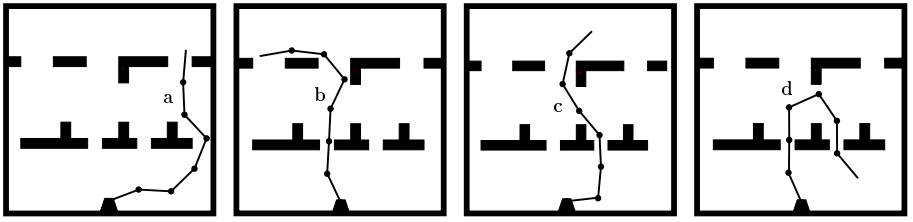
checking routines are used, as well as a robot-specific local planner. Furthermore, "difficult" nodes are heuristically identified during the roadmap construction phase, and "expanded" subsequently. We see that roughly 1 minute was sufficient to obtain roadmaps solving the 6 queries. These roadmaps consist of approximately 4000 nodes.

4 Application to nonholonomic robots

In this section we deal with nonholonomic mobile robots. More specifically, we apply *PPP* to car-like robots and tractor-trailer robots. We consider two types of car-like robots, i.e., such that can drive both forwards and backwards, and such that can only drive forwards. We refer to the former as *general car-like robots*, and to the latter as *forward car-like robots*. First however we give a brief overview on previous work on nonholonomic motion planning.

4.1 Some previous work on nonholonomic motion planning

Nonholonomic constraints add an extra level of difficulty to the path planning problem. The paths must (1) be collision free and (2) describe motions that are executable for the robot. We refer to such paths as *feasible* paths.



	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%

Fig. 4. A seven dof articulated robot in a very constrained environment and the query test set.

For *locally controllable robots* [6], the existence of a feasible path between two configurations is equivalent to the existence of a collision free path, due to the fact that for any collision free path there exists a feasible path lying arbitrarily close to it. This fundamental property has led to a family of algorithms, decomposing the search in two phases. They first try to solve the geometric problem (i.e., the problem for the holonomic robot that is geometrically equivalent to the nonholonomic one). Then they use the obtained collision-free path to build a feasible one. So in the first phase the decision problem is solved, and only in the second phase are the nonholonomic constraints taken into account. One such approach was developed for car-like robots [26], using Reeds and Shepp works on optimal control to approximate the geometric path. In [34] Reeds and Shepp presented a finite family of paths composed of line segments and circle arcs containing a length-optimal path linking any two configurations (in absence of obstacles). The planner introduced in [26] replaces the collision-free geometric path by a sequence of Reeds and Shepp paths. This complete and fast planner was extended to the case of tractor-trailer robots, using near optimal paths numerically computed [27,12] (so far the exact optimal paths for the tractor-trailer system in absence of obstacle are unknown). The resulting planners are however neither complete nor time-efficient. The same scheme was used for systems that can be put into the *chained form*. For these sys-

tems, Tilbury et al. [50] proposed different controls to steer the system from one configuration to another, in absence of obstacles. Sekhavat and Laumond prove in [38] that the *sinusoidal inputs* proposed by Tilbury et al. can be used in a complete algorithm transforming any collision-free path to a feasible one. This algorithm was implemented for a car-like robot towing one or two trailers, which can be put into the chained form, and finds paths in reasonable times [38]. A multi-level extension of this approach has been presented in [40] which further improves the running times of this scheme by separating the nonholonomic constraints mutually, and introducing separately. The scheme is however, as pointed out, only applicable to locally controllable robots. For example, forward car-like robots do not fall in this class.

Barraquand and Latombe [6] have proposed a heuristic brute-force approach to motion planning for nonholonomic robots. It consists of heuristically building and searching a graph whose nodes are small axis-parallel cells in C-space. Two such cells are connected in the graph if there exists a basic path between two particular configurations in the respective cells. The completeness of this algorithm is guaranteed up to appropriate choice of certain parameters, and it does not require local controllability of the robot. The main drawback of this planner is that when the heuristics fail it requires an exhaustive search in the discretised C-space. Furthermore, only the cell containing the goal configuration is reached, not the goal configuration itself. Hence the planner is inexact. Nevertheless, in many cases the method produces nice paths (with minimum number of reversals) for car-like robots and tractors pulling one trailer. For systems of higher dimension however it becomes too time consuming. Ferbach [11] builds on the approach of Barraquand and Latombe method in his *progressive constraints* algorithm in order to solve the problem in higher dimensions. First a geometric path is computed. Then the nonholonomic constraints are introduced progressively in an iterative algorithm. Each iteration consists of exploring a neighbourhood of the path computed in the previous iteration, searching for a path that satisfies more accurate constraints. Smooth collision-free paths in non-trivial environments were obtained with this method for car-like robots towing two and three trailers. The algorithm however does not satisfy any form of completeness.

The probabilistic path planner *PPP* has been applied to various types of nonholonomic robots. An advantage over the above single shot methods is the fact that a roadmap is constructed just ones, from which paths can subsequently be retrieved quasi-instantaneously. Also, local robot controllability is not required. A critical point of *PPP* when applied to nonholonomic robots is however the speed of the nonholonomic local planner. For car-like robots very fast local planners have been developed. Thanks to this, *PPP* applied to the car-like robots resulted in fast and probabilistically complete planners for car-like robots that move both forwards and backwards, as well as for such

that can only move forwards [45,47]. Local planners for tractor-trailer robots however tend to be much more time-consuming, which makes direct use of *PPP* less attractive. In [49] a local planner is presented and integrated into *PPP*, that uses exact closed form solutions for the kinematic parameters of a tractor-trailer robot. In [39] the local planner using sinusoidal inputs for chained form systems is used. For robots pulling more than one trailer, this local planner appeared to be too expensive for capturing the connectivity of the free C-space. For this reason, and inspired by the earlier mentioned works [26,27,12,38], in [39] a two-level scheme is proposed, where at the first level the portion of CS_{free} is reduced to a neighbourhood of a geometric path, and at the second level a (real) solution is searched for within this neighbourhood (by *PPP*). The multi-level algorithm proposed in [40] can in fact been seen as a generalisation of this two level scheme.

4.2 Description of the car-like and tractor-trailer robots

We model a car-like 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 car-like 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_{max} is the (symmetric) joint bound. The car-like part (the *tractor*) is a car-like robot. The extra part (the *trailer*) is subject to further nonholonomic constraints. Its motions are (physically) dictated by the motions of the tractor (For details, see for example [25,40]).

For car-like robots, the paths constructed will be sequences of translational paths (describing straight motions) and rotational paths (describing motions of constant non-zero curvature) only. It is a well-known fact [25] that if for a (general or forward) car-like robot a feasible path in the open free C-space exists between two configurations, then there also exists one that 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.

4.3 Application to general car-like robots

We now apply *PPP*, using an undirected graph, to general car-like robots. This again asks for filling in some of the (robot specific) details that have been left open in the discussion of the general method.

Filling in the details

The local planner: A *RTR path* is defined as the concatenation of a rotational path, a translational path, and another rotational path. Or, in other words, it is the concatenation of two circular arcs and a straight line segment, with the latter in the middle. The *RTR local planner* constructs the shortest *RTR* path connecting its argument configurations. 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 [45] for more details). Furthermore, the RTR local planner satisfies a local topological property that guarantees probabilistic completeness (See Section 5).

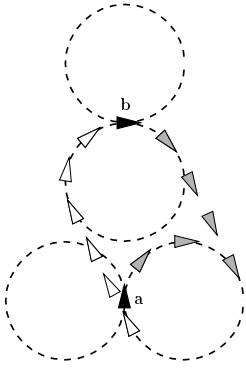


Fig. 5. Two RTR paths for a triangular car-like robot, connecting configurations a and b.

An alternative to the RTR local planner is to use a local planner that constructs the shortest (car-like) path connecting its argument configurations [34], [42]. Constructing shortest car-like 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.
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 also called *generalised polygons*) 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 one single intersection test for a polygon.

The distance measure: We use a distance measure that 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 distance measure as the *RTR distance measure*, and we denote it by D_{RTR}.

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

Let c_s be the start configuration of a random 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 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.

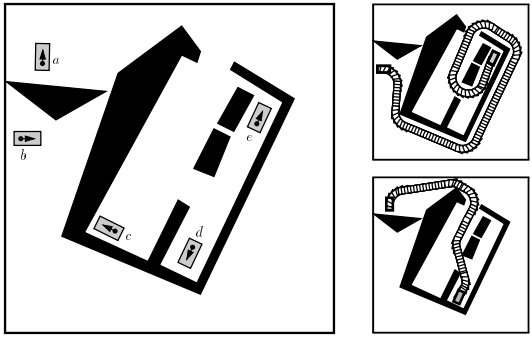
Good values for N_W and L_W must be experimentally derived (the values we use are given in the next section).

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 [47] for more details). Furthermore, as for free-flying robots, we use the (run-time) structure of the graph G in order to guide the node generation.

Simulation results We have implemented the planner as described above, and some simulation 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 simulation results are presented in the same form as for the holonomic robots in Section 3. That is, for different roadmap construction times we count how often graphs are obtained that solve particular, predefined, queries.

Figure 6 shows a relatively easy scene, together with the robot $\mathcal{A}$ positioned at a set of configurations $\{a, b, c, d, e\}$. The topology is simple and there are only a few narrow passages. We use $\{(a, b), (a, d), (b, e), (c, e), (d, e)\}$ as query test set T_Q. (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 neighbourhood size *maxdist* is 0.5. We see that after only 0.3 seconds of roadmap construction, the networks solve each of the queries in most cases (but not all). Half a second of construction is sufficient for solving each of the queries, in all 20 trials. The corresponding roadmaps contain about 150 nodes.

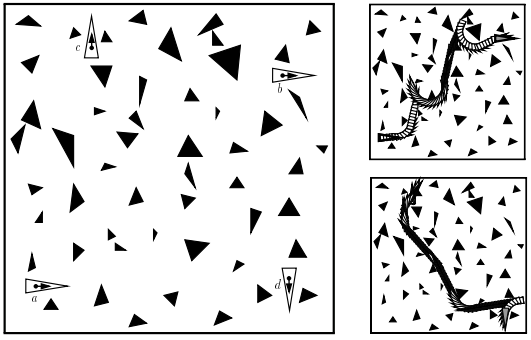


	0.1 sec.	0.2 sec.	0.3 sec.	0.4 sec.	0.5 sec.
(a,b)	20%	90%	100%	100%	100%
(a,d)	35%	55%	85%	95%	100%
(b,e)	15%	75%	85%	100%	100%
(c,e)	60%	90%	100%	100%	100%
(d,e)	50%	85%	95%	100%	100%

Fig. 6. A simple scene. At the top right, two paths computed by the planner and smoothed in 1 second are shown.

Figure 7 (again together with a robot $\mathcal{A}$ placed at different configurations $\{a, b, c, d\}$), shows a completely different type of scene. It contains many (small) obstacles and is not at all "corridor-like". Although many individual path planning problems in this scene are quite simple, the topology of the free C-space is quite complicated, and can only be captured well with 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 the queries (a, b) and (c, d)). We see that about 2 seconds of construction are required to obtain

roadmaps that are (almost) guaranteed to solve each of the queries. Their number of nodes is about 350.



	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%

Fig. 7. A more complicated scene, and its test configurations. At the top right, two paths computed by the planner and smoothed in 1 second are shown.

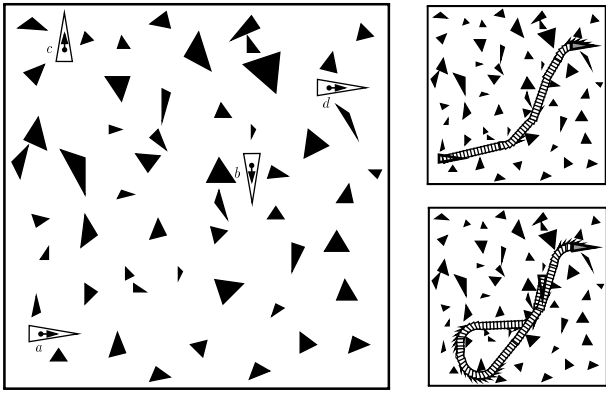
4.4 Application to forward car-like robots

Forward car-like robots, as pointed out before, are not $\mathcal{C}$-symmetric. 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 roadmap construction algorithm we refer to [32].

The robot specific components, such as the local planner, the metric, and the random walks are quite similar to those used for general car-like robots, as described in Section 4.3. The local planner constructs the shortest *forward RTR path* connecting its argument configurations. A forward RTR path 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 car-like 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).

Simulation results In Figure 8 we give some results for the same scene as Figure 7. We see that the queries are most likely to be solved after 5 seconds of roadmap construction, and (almost) surely after 7.5 seconds, by roadmaps consisting of around 700 nodes. This means that about four times more time is required than for general car-like robots.



	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%

Fig. 8. Motion planning for a forward car-like robot.

4.5 Application to tractor-trailer robots

As last example of nonholonomic robots, we now (briefly) consider tractor-trailer robots, and well such that can drive both forwards and backwards. These robots have symmetrical control systems and, hence, undirected underlying graphs are sufficient. We will not go into many details. We refer to [39,40]

for a more thorough discussion of the topic. We use a local planner, by Sekhavat and Laumond [38], that 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 that guarantees probabilistic completeness of the global planner. As distance measure we use (cheap) approximations of the workspace lengths of the local paths. The random walks in the query phase are basically as those for general car-like robots, except that the trailers orientation must be kept track of during each motion of the tractor. This can be done using exact closed form solutions for the kinematic parameters of tractor-trailer robots under constant curvature motions of the tractor [49]. 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.

Simulation 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).

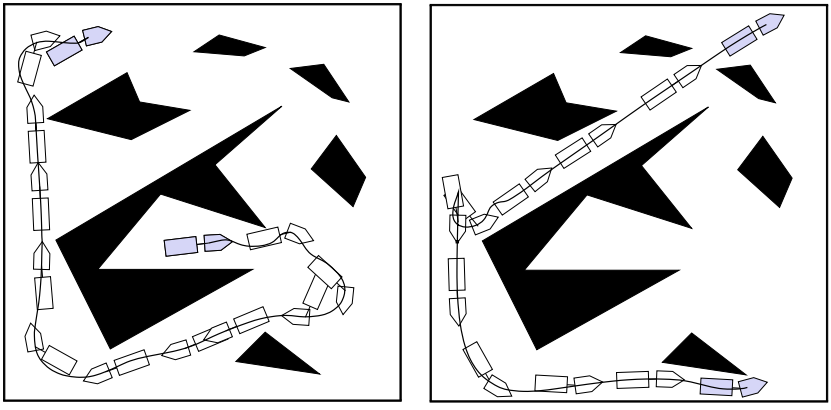


Fig. 9. Two feasible paths for a tractor-trailer robot, obtained in 10 seconds.

5 On probabilistic completeness of probabilistic path planning

In this section we discuss some aspects regarding *probabilistic completeness* of *PPP*, and we prove this completeness for the specific planners described in this

chapter. We will assume a slightly simplified version of the planning scheme. Instead of trying to connect the start configuration s and goal configuration g to the graph with some graph retractions (as described in Section 2.2), we simply add s and g to the graph as (initial) nodes. A query consists of just a graph search. This simplification of the query phase is for ease of presentation. All results presented in this section directly hold for the case where queries are performed as described earlier (in Section 2.2), using graph retractions and random walks.

A path planner is called *probabilistically complete* if, given any problem that is solvable in the open free C-space, the probability that the planner solves the problem goes to 1 as the running time goes to infinity. Hence, a probabilistically complete path planner is guaranteed to solve such a problem, provided that it is executed for a sufficient amount of time. For ease of presentation we introduce some shorthand notations. We denote the version of *PPP* 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. We say L is *symmetric* iff, for arbitrary configurations a and b, $L(a, b)$ equals $L(b, a)$ reversed. Furthermore, we again denote the C-space, respectively the free C-space, by $\mathcal{C}$, respectively $\mathcal{C}_f$.

We point out that with *PPP* one obtains a probabilistically complete planner for any robot that is *locally controllable* (see below), provided that one defines the local planner properly. If, in addition to the 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 local controllability of the robot, a local planner 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 locally controllable robots with symmetric control systems. All holonomic robots, as well as for example general car-like robots and tractor-trailer robots, fall into this class. Forward car-like robots however are not locally controllable (and neither symmetric). In Section 5.2 we show that all the planners described in this chapter are probabilistically complete.

First we define the concept local controllability (in the literature also referred to as small-time local controllability or local-local controllability), adopting the terminology introduced by Sussman [43]. 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 *locally controllable* iff for any configuration $c \in \mathcal{C}$ $A_{\Sigma_{\mathcal{A}}}(\leq T, c)$ contains a neighbourhood of c (or, equivalently, a ball centred at c) for all $T > 0$. It is a well-known

fact that, given a configuration c, the area a locally controllable robot $\mathcal{A}$ can reach without leaving the ϵ-ball around c (for any $\epsilon > 0$) is the entire open ϵ-ball around c.

5.1 The general local topology property

We assume now that robot $\mathcal{A}$ is locally controllable. For probabilistic completeness of *PPP* a local planner L is required that exploits the local controllability of $\mathcal{A}$. This will be 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 centred at configuration c by $B_\epsilon(c)$.

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 $B_\delta(c)$ 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 local controllability. Theorem 5.1 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 5.1. *If L is a local planner verifying the GLT-property, then* PPP(L) *is probabilistically complete.*

Proof. The theorem can be proven 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 that 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 $\forall c \in \mathcal{C} : B_\delta(c) \subset R_{L,\frac{3}{4}\epsilon}(c)$. Then, consider a covering of P by balls $B_1, \ldots, B_k$ of radius $\frac{1}{4}\delta$, such that balls B_i and B_{i+1}, for $i \in \{1, \ldots, k-1\}$, partially overlap. Assume each such ball B_i contains a node v_i of G. Then, $|v_i - v_{i+1}| \leq \delta$, and each node v_i has a C-space

clearance of at least $\epsilon - \frac{1}{4}\delta \geq \frac{3}{4}\epsilon$ (since $\delta \leq \epsilon$). Hence, due to the definition of δ, we have

$$L(v_i, v_{i+1}) \subset B_{\frac{3}{4}\epsilon}(v_i) \subset \mathcal{C}_f$$

It follows that if all the balls $B_1, \ldots, B_k$ contain a node of G, s and g will be graph-connected. Since, due to the random node adding, this is guaranteed to be the case within a finite amount of time, this concludes the proof. See also Figure 10. ■

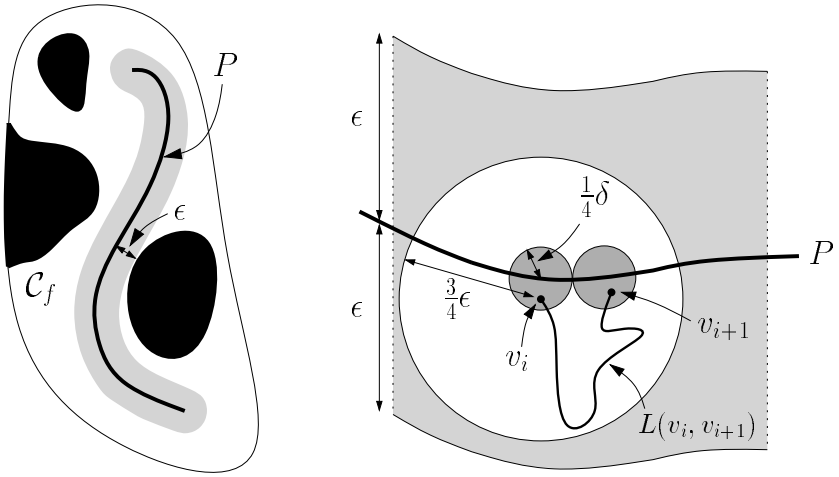


Fig. 10. Path P has clearance $\epsilon > 0$. If $\delta > 0$ is chosen such that $\forall c \in \mathcal{C}$: $B_\delta(c) \subset R_{L,\frac{3}{4}\epsilon}(c)$, then we see that nodes in overlapping $\frac{1}{4}\delta$-balls, centred at configurations of P, can always be connected by the local planner.

Clearly, given a locally controllable robot, the GLT-property is a proper criterion for choosing the local planner (sufficient conditions for local controllability of a robot are given in, e.g., [44]). Path planning among obstacles for car-like robots using local planners with the GLT-property has also been studied by Laumond [28,18].

For 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 centred 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 planners verifying it, we refer to [47].

5.2 Probabilistic completeness with the used local planners

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

Locally controllable robots The general holonomic local planner L for holonomic robots 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, L verifies the GLT-property.

Theorem 5.2. $\mathrm{PPP}_u(L)$*, with* L *being the general holonomic local planner, is probabilistically complete for all holonomic robots.*

The planner for general car-like robots uses the RTR local planner. One can prove that this planner verifies the LTP-property [47]. Again, as stated in the following theorem, this guarantees probabilistic completeness.

Theorem 5.3. $\mathrm{PPP}_u(L)$*, with* L *being the RTR local planner, is probabilistically complete for general car-like robots.*

Regarding tractor-trailer robots, Sekhavat and Laumond prove in [38] 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.

Theorem 5.4. $\mathrm{PPP}_u(L)$*, with* L *being the sinusoidal local planner, is probabilistically complete for tractor-trailer robots (with arbitrary number of trailers).*

Forward car-like robots As pointed out before, the theory of the previous sections applies only to robots that are locally controllable. If a robot does not have this property, a local planner verifying the GLT-property will 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 car-like robots are not 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, which does not directly generalise to other cases, uses a property of RTR forward paths stated in Lemma 5.5.

Lemma 5.5. *Let L be the RTR forward local planner, and let Q be a RTR forward path connecting configurations a and b with a straight line path of non-zero length and both arc paths of total curvature less than $\frac{1}{2}\pi$. Then:*

$$\forall \epsilon > 0 : \exists \delta > 0 : \forall (\tilde{a}, \tilde{b}) \in B_\delta(a) \times B_\delta(b) : L(\tilde{a}, \tilde{b}) \text{ lies within distance } \epsilon \text{ of } Q \ [1]$$

Theorem 5.6. $\mathrm{PPP}_d(L)$*, with L being the RTR forward local planner, is probabilistically complete for forward car-like robots.*

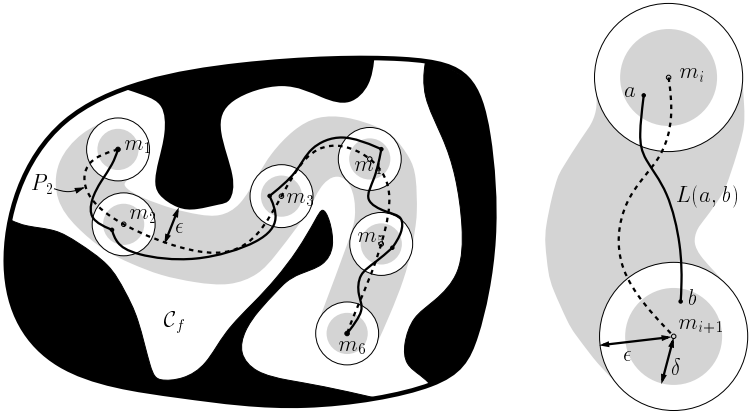


Fig. 11. This figure illustrates the proof of Theorem 5.6. P_2 is a path, feasible for a forward car-like robot, of clearance $\epsilon > 0$. Centred at the configurations m_i are balls B_i of a radius $\delta > 0$, such that any pair of configurations $(a, b) \in B_i \times B_{i+1}$ is connected by the RTR forward local planner L with a path lying within distance ϵ of P_2, and hence lying in $\mathcal{C}_f$.

We give only a sketch of the proof here (See also Figure 11). 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, that is feasible for our forward car-like 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, that consists of (a finite

number of) straight line segments and circular arcs, such that no two distinct arcs are adjacent and each arc has a total curvature of less than $\frac{1}{2}\pi$.[2]

Assume k is the number of arcs in P_2. Let $m_1 = s$, $m_k = g$, and $\{m_2, \ldots, m_{k-1}\}$ be points on P_2 such that m_i is the midpoint of the i-th arc of P_2 (that is, the unique point on the arc with equal distance to both end-points). Clearly, m_i is connected m_{i+1} by a forward RTR path with a straight line segment of non-zero length and both arc paths of total curvature less than $\frac{1}{2}\pi$ (for all $j \in \{1, \ldots, k-1\}$).

Let $\epsilon > 0$ be the clearance of P_2, and take $\delta > 0$ such that, for all $j \in \{1, \ldots, k-1\}$:

$$\forall (a, b) \in B_\delta(m_j) \times B_\delta(m_{j+1}) : L(a, b) \text{ lies within distance } \epsilon \text{ of } Q$$

It follows from Lemma 5.5 that such a $\delta > 0$ always exists. When a node of G is present in every ball $B_\delta(m_j)$ for $2 \leq j < k$, G will contain a path connecting s to g. We know, due to the probabilistic nature of the node adding, that the probability of obtaining such a graph grows to 1 when the roadmap construction time goes to infinity.

6 On the expected complexity of probabilistic path planning

In the previous section we have formulated properties of local planners that guarantee probabilistic completeness of *PPP* for locally controllable robots. If these properties are satisfied, we know that as the running time of *PPP* goes to infinity, the probability of solving any solvable problem goes to 1. However, nothing formal has yet been said about (expected) convergence times of the algorithm. In practice, one will not be satisfied with the guarantee that "eventually a path will be found". For real life applications, some estimate of the running time beforehand is desirable.

Simulation results obtained by the application of *PPP* on certain "typical" problems can increase our trust in the planners performance and robustness, but they do not describe a formal relation between the probabilities of failure and running times in general, and neither do they provide a theoretical explanation for the empirically observed success of the probabilistic planner. Recently some first theoretical results on expected running times of probabilistic planners have been obtained.

Kavraki et al. [22,20,3] show that, under certain geometric assumptions about the free C-space $\mathcal{C}_f$, it is possible to establish a relation between the

[2] 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.

probability that probabilistic planners like *PPP*[3] find paths solving particular problems, and their running times. They suggest two such assumptions, i.e., the *visibility volume assumption* and the *path clearance assumption*. We will discuss these assumptions and present the main results obtained by Kavraki et al.. Also, we will discuss to what extent these results hold for nonholonomic robots, and, where possible, we will adapt them appropriately. Furthermore, we introduce a new assumption on configuration space, the *ϵ-complexity assumption*, under which it is possible to relate the success probabilities and running times of *PPP* to the complexity of the problems that are to be solved.

Throughout this section we use the notations introduced in the previous section, and, unless stated otherwise, we will assume that *PPP* with undirected underlying graphs (i.e., PPP_u) is used.

6.1 The visibility volume assumption

The visibility volume assumption uses a notion of "visibility" defined by the used local planner. A free configuration a is said to "see" a free configuration b if the local path $L(a, b)$ lies entirely in $\mathcal{C}_f$. The visibility volume assumption now states that every free configuration "sees" a subset of $\mathcal{C}_f$ whose volume is at least an ϵ fraction of the total volume of $\mathcal{C}_f$. If this holds, $\mathcal{C}_f$ is called *ϵ-good*.

The analyses assumes a somewhat more complex planner than *PPP* as defined in Section 2. It differs from *PPP* in that after a probabilistic roadmap $G = (V, E)$ has been constructed (by the roadmap construction algorithm in Section 2.1), an extra post-processing step is performed, referred to by the authors as *Permeation*. Permeation assumes the existence of a complete planner, that is, a planner solving any solvable problem, and returning failure for any non-solvable one. What permeation does, is invoking the complete planner for certain pairs $(a, b) \in V \times V$ that are not graph connected. Planners based on this scheme have been implemented by Kavraki et al. (E.g., [21]). However, instead of a complete planner (which, in general, is not available) the potential field planner *RPP* has been utilised. Since *RPP* is only probabilistically complete, the mentioned planners are merely approximations of the algorithm sketched above.

Due to the assumed completeness of the invoked planner, provided that the complete planner is invoked for enough pairs of nodes in V, permeation leads to a roadmap where every connected component of $\mathcal{C}_f$ contains at most one connected component of the roadmap G. For convenience, we will say that such roadmaps have *perfect connectivity*.

Let us now assume that G_P has such perfect connectivity. Then, if both s and g "see" a node of G_P, the planner will return a path solving this problem

[3] Kavraki et al. refer to *PPP* by the name *PRM* (Probabilistic RoadMap planner).

if it is solvable, and return failure otherwise. In other words, on the portion of C_f that "sees" some node of the roadmap, the planner is complete. Note that during the permeation step, no nodes are added to G. Hence, the question whether a solvable query will be answered correctly depends solely on the set of nodes V in G. Theorem 6.1 addresses this question. V is called *adequate* if the portion of $\mathcal{C}_f$, not visible from any $c \in V$, has a volume of at most $\frac{1}{2}\epsilon \cdot \text{Volume}(\mathcal{C}_f)$. The theorem gives a lower bound for the probability of V being adequate.

Theorem 6.1. ***(Kavraki et al. [22], Barraquand et al. [3])*** *Assume $G = (V, E)$ is a roadmap generated by* PPP *in a free C-space that is ϵ-good. Let $\beta \in (0, 1]$ be a real constant, and let C be a positive constant large enough such that $\forall x \in (0, 1] : (1-x)^{(\frac{C}{x} \log \frac{1}{x})} \leq x\frac{\beta}{4}$ Now, if $|V| \geq \frac{C}{\epsilon} \log \frac{1}{\epsilon}$, then V is adequate with probability at least $1 - \beta$.*

Regarding the complexity of the roadmap construction, one must estimate the number of calls to the complete planner during the permeation step, required for obtaining a roadmap G_P of perfect connectivity. Theorem 6.2 provides such an estimate.

Theorem 6.2. ***(Kavraki et al. [22], Barraquand et al. [3])*** *Let $w_1 \geq w_2 \geq \cdots \geq w_k$ be the sizes of the connected components of the roadmap $G = (V, E)$. There exists a (randomised) algorithm that extends G to a roadmap G_P of perfect connectivity, whose expected number of calls of the complete planner is at most:*

$$2 \sum_{i=1}^{k} i \cdot w_i - |V| - k$$

Furthermore, with high probability, the number of calls is at most:

$$O\left(\sum_{i=1}^{k} i \cdot w_i \cdot \log(|V|)\right)$$

Symmetry of the local planner L is required and assumed in the above. However, apart from this, the ϵ-goodness assumption is very general, and the given analysis is not only valid for holonomic robots (on which the authors focus), but also for nonholonomic ones.

However, the question arises in how far the theoretical results are "practical" for nonholonomic robots, since the "visibility" induced by a nonholonomic local planner will be a quite hard to grasp concept that can hardly be regarded as a geometric property of $\mathcal{C}_f$. There does not seem to be much hope that we will ever be able to measure the ϵ-goodness of $\mathcal{C}_f$ for nonholonomic robots, in non-trivial cases (insofar as there is such hope for holonomic ones).

6.2 The path clearance assumption

In the path clearance assumption, there exists a collision-free path $\mathcal{P}$ between the start configuration s and goal configuration g, that has some clearance $\epsilon > 0$ with the C-space obstacles. Throughout this section we denote the volume of an object A by $\mathcal{V}(A)$. In [20], Kavraki et al. study the dependence of the failure probability of *PPP* (the normal version) to connect s and g on (1) the length of $\mathcal{P}$, (2) the clearance ϵ, and (3) the number of nodes in the probabilistic roadmap G. Their main result is described by the following theorem:

Theorem 6.3. ***(Kavraki et al. [20], Barraquand et al. [3])*** *Let $\mathcal{A}$ be a holonomic robot, L be the general holonomic local planner (for $\mathcal{A}$), and $G = (V, E)$ be a graph constructed by* PPP(L). *Assume configurations s and g are connectable by a path $\mathcal{P}$ of length λ, that has a clearance $\epsilon > 0$ with the (C-space) obstacles. Let $\alpha \in (0, 1]$ be a real constant, and let a be the constant $\frac{1}{2^n}\mathcal{V}(\mathcal{B}_1)/\mathcal{V}(\mathcal{C}_f)$, where $\mathcal{B}_1$ denotes the unit ball in the C-space $\mathbb{R}^n$. Now if $|V|$ is such that*

$$\frac{2\lambda}{\epsilon}(1 - a\epsilon^n)^{|V|} \leq \alpha$$

then, with probability at least $1 - \alpha$, s and g will be graph-connected in G (See also Figure 12).

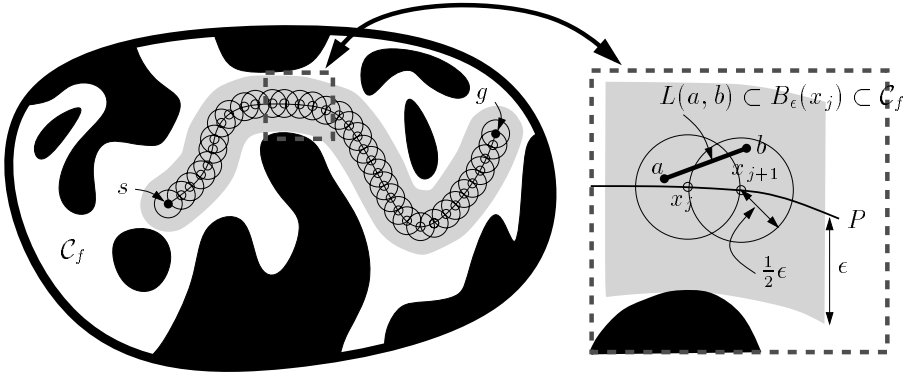


Fig. 12. We see that configuration s is connectable to configuration g by a P of clearance ϵ. Let $x_0 = s, x_1, \ldots, x_k = g$ be points on P, such that $|x_j - x_{j+1}| \leq \frac{1}{2}\epsilon$, for all j. If each ball $B_{\frac{1}{2}\epsilon}(x_j)$ contains a node of G, then s and g will be graph-connected.

The proof of this theorem is quite straightforward. Given a path P of clearance $\epsilon > 0$, one can consider a covering of P by balls of radius $\frac{1}{2}\epsilon$ as shown in Figure 12, and bound the probability that one of these balls contains no node of G. Since, if each of these balls does contain a node, G is guaranteed to contain a path connecting the start and goal configuration, this gives an upper bound for the failure probability of *PPP*.

A number of important facts are implied by Theorem 6.3. E.g., the number of nodes required to be generated, in order for the planner to succeed with probability at least $1 - \alpha$, is logarithmic in $\frac{1}{\alpha}$ and λ, and polynomial in $\frac{1}{\epsilon}$. Furthermore, the failure probability α is linear in the path length λ.

The analyses assumes the use of the general holonomic local planner (as described in Section 3.1). Hence it is assumed that the robot is holonomic. An underlying assumption is namely that the ϵ-reachable area of any configuration c consists of the entire ϵ-ball $B_\epsilon(\mathrm{c})$, surrounding c. From theoretical point of view, as pointed out in the previous section, for any locally controllable robot a local planner exists for which the ϵ-reachable area of any configuration c consists of the entire open ϵ-ball centred at c. Such a local planner would allow for the result assuming the path clearance to be directly applied to such robots. However, it is not realistic to assume the "ϵ-ball reachability" for a nonholonomic local planner, since for most robots we are not able to construct such local planners, and, if we could, they would probably be vastly outperformed (in terms of computation time) by simple local planners verifying only weaker (but sufficient) topological properties, such as those presented in the previous section. However, the analyses presented in [20] can be extended to the case where the local planner verifies only the GLT-property. Through this, we can give running time estimates for locally controllable nonholonomic robots that are realistic in the sense that we can actually build the planners that we analyse. Corollary 6.4 extends the result of Theorem 6.3 to locally controllable nonholonomic robots with local planners verifying the *GLT*-property.

Corollary 6.4. *Let $\mathcal{A}$ be a fully controllable robot, L be a local planner for $\mathcal{A}$ verifying the* GLT*-property, and $G = (V, E)$ be a graph constructed by* PPP(L). *Assume configurations s and g are connectable by a path $\mathcal{P}$ of length λ, that has a clearance $\epsilon > 0$ with the (configuration space) obstacles. Take $\delta > 0$ such that*

$$\forall c \in \mathcal{C} : B_\delta(c) \subset R_{L,\frac{3}{4}\epsilon}(c)$$

Let $\alpha \in (0, 1]$ be a real constant, and $\mathcal{V}_1$ be the volume of the unit ball in the C-space $\mathbb{R}^n$. Now if $|V|$ is such that

$$\frac{2\lambda}{\delta}\left(1 - \frac{\mathcal{V}_1}{4^n \mathcal{V}(\mathcal{C}_f)}\delta^n\right)^{|V|} \leq \alpha$$

then, with probability at least $1 - \alpha$, s and g will be graph-connected in G.

Since, by definition of the *GLT*-property, δ is a constant with respect to ϵ, the dependencies implied by Theorem 6.3 hold for nonholonomic robots as well.

6.3 The ϵ-complexity assumption

A drawback of Theorem 6.3 and the above corollary is that no relation is established between the failure probability and the *complexity* of a particular problem. In our opinion, to a considerable extent, the observed success of *PPP* lies in the fact that not the complexity of the C-space, but the complexity of the resulting path defines the (expected) running time of *PPP*. For example, assume a particular problem is solvable by a path P of clearance $\epsilon > 0$, consisting of say 4 straight line segments. Consider three balls of radius ϵ, centred at the 3 inner nodes of P. Then, as is illustrated in Figure 13, it suffices that all the 3 balls contain a node of G to guarantee that the problem is solved. We see in this example that the failure probability in no way relates to the length of the path, and neither to the complexity of $\mathcal{C}_f$. The only relevant factors are the clearance and the complexity of the path. Definition 4 introduces the notion of ϵ-*complexity*, which captures this measure of problem complexity. We refer here to a path composed of k straight line segments as a piecewise linear path of complexity k.

Definition 4. *Given a holonomic robot and a particular path planning problem (s, g), let P be the lowest complexity piecewise-linear path connecting s and g, that has a C-space clearance of $\epsilon > 0$. We define the* ϵ-complexity *of problem (s, g) as the complexity of P.*

Theorem 6.5 gives a result relating the failure probability of *PPP* to the ϵ-*complexity* of the problem to be solved. It applies only to holonomic robots and assumes the use of the general holonomic local planner.

Theorem 6.5. *Let $\mathcal{A}$ be a holonomic robot, L be the general holonomic local planner (for $\mathcal{A}$), and $G = (V, E)$ be a graph constructed by* PPP(L). *Assume (s, g) is a problem of ϵ-complexity ζ. Let $\alpha \in (0, 1]$ be a real constant, and $\mathcal{V}_1$ be the volume of the unit ball in the C-space $\mathbb{R}^n$. Now if $|V|$ is such that*

$$(\zeta - 1)\left(1 - \frac{\mathcal{V}_1}{\mathcal{V}(\mathcal{C}_f)}\epsilon^n\right)^{|V|} \leq \alpha$$

then, with probability at least $1 - \alpha$, s and g will be graph-connected in G.

This theorem can be proven quite easily. Given a problem of ϵ-complexity ζ, there exists a piecewise linear path P of complexity ζ and clearance $\epsilon > 0$

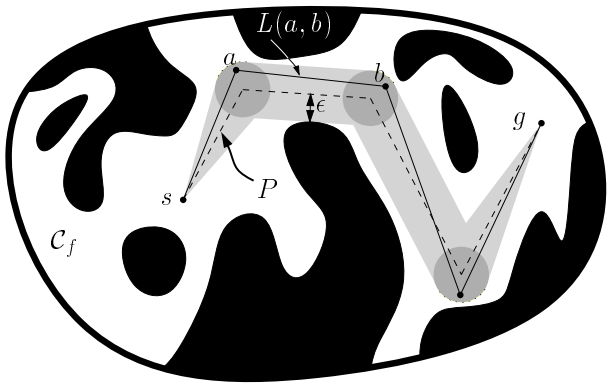


Fig. 13. We see that configuration s is connectable to configuration g by a piecewise linear path P (dashed) of complexity 4 and clearance ϵ. If each of 3 dark grey balls (of radius ϵ, placed at the vertices of P) contains a node of G, then the G contains a path, lying in the grey area, that connects s and g.

solving it. We can place balls of radius ϵ at the vertices of P, and bound the probability the one of these balls contains no node of G. Since, if each of these balls does contain a node, G is guaranteed to contain a path connecting the start and goal configuration, this gives us an upper bound for the failure probability of *PPP*.

So we now also have a linear dependence of the failure probability, and a logarithmic dependence of $|V|$, on the *complexity* ζ of the path P, that is, on the ϵ-complexity of the problem. We note that the existence of a path of a certain clearance $\epsilon > 0$ and implies the existence of a piecewise linear path of a similar clearance.

7 A multi-robot extension

We conclude this chapter with an extension of *PPP* for solving multi-robot path planning problems. That is, problems involving a number of robots, present in the same workspace, that are to change their positions while avoiding (mutual) collisions. Important contributions on multi-robot path planning include [37,13,10,51,41,8,30,29,4,5,17,2,52]. For overviews we refer to [25] and [15].

Most previous successful planners fall into the class of *decoupled planners*, that is, planners that first plan separate paths for the individual robots more or less independently, and only in a later stage, in case of collisions, try to adapt the paths locally to prevent the collisions. This however inherently leads to planners that are not complete, that is, that can lead to deadlocks. To

obtain some form of completeness, one must consider the separate robots as one composite system, and perform the planning for this entire system. However, this tends to be very expensive, since the composite C-space is typically of high dimension, and the constraints of all separate add up.

For example, multi-robot problems could be tackled by direct application of *PPP*. The robot considered would be composed of the separate "simple" robots, and the local planner would construct paths for this composite robot. This is a very simple way of obtaining (probabilistically complete) multi-robot planners. However, as mentioned above, a drawback is the high dimension of the configuration space, which, in non-trivial scenes, will force *PPP* to construct very large roadmaps for capturing the structure of $\mathbf{C}_f$. Moreover, each local path in such a roadmap will consists of a number of local paths for the simple robots, causing the collision checking to be rather expensive.

In this section we describe a scheme where a roadmap for the composite robot is constructed only after a discretisation step that allows for disregarding the actual C-space of the composite robot. See Figure 14 for an example of a multi-robot path planning problem, and a solution to it, computed by a planner based on the scheme.

We will refer to the separate robots $\mathcal{A}_1, \ldots, \mathcal{A}_n$ as the *simple robots*. One can also consider the simple robots together to be one robot (with many degrees of freedom), the so-called *composite robot*. A feasible path for the composite robot will be referred to as a *coordinated path*. We assume in this paper that the simple robots are identical, although, with minor adaptions, the presented concepts are applicable to problems involving non-identical robots as well.

A roadmap for the composite robot is constructed in two steps. First, a *simple roadmap* is constructed for just one robot with *PPP*. Then n of such roadmaps are combined into a roadmap for the composite robot (consisting of n simple robots). We will refer to such a composite roadmap as a *super-graph*. After such a super-graph has been constructed, which needs to be done just once for a given static environment, it can be used for retrieval of coordinated paths. We will present two super-graph structures: *flat super-graphs* and *multi-level super-graphs*. The latter are a generalisation of flat super-graphs, that consume much less memory for problems involving more than 3 robots.

The scheme is a flexible one, in the sense that it is easily applicable to various robot types, provided that one is able to construct simple roadmaps for one such robot. Furthermore, proper construction of the simple roadmaps guarantees *probabilistic completeness* of the resulting multi-robot planners [46]. In this paper we apply the super-graph approach to *car-like robots*. We give simulation results for problems involving up to 5 robots moving in the same constrained environment.

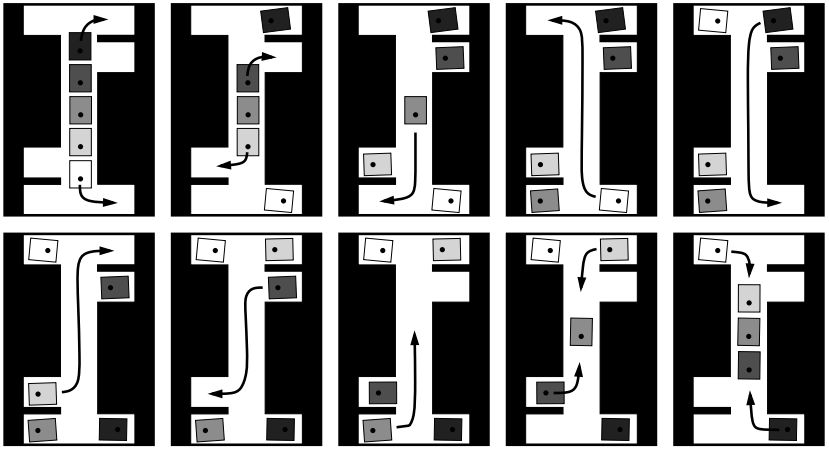


Fig. 14. An example of a multi-robot path planning problem, with a solution shown (generated by the multi-level super-graph planner). Five car-like robots are in a narrow corridor, and they are to reverse their order.

7.1 Discretisation of the multi-robot planning problem

The first step of our multi-robot planning scheme consists of computing a simple roadmap, that is, a roadmap for the simple robot $\mathcal{A}$. We assume that this roadmap is stored as a graph $G = (V, E)$, with the nodes V corresponding to collision-free configurations, and the edges E to feasible paths, also referred to as *local paths*. We say a node *blocks* a local path, if the volume occupied by $\mathcal{A}$ when placed at the node intersects the volume swept by $\mathcal{A}$ when moving along the local path. Basically, any algorithm that constructs roadmaps can be used in this phase. We will use *PPP*.

Given a graph $G = (V, E)$ storing a simple roadmap for robot $\mathcal{A}$, we are interested in solving multi-robot problems using G. We assume here that the start and goal configurations of the simple robots are present as nodes in G (otherwise they can easily be added). 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, and we also disallow motions along local paths that are blocked by the nodes at which the other robots are stationary. We refer to such paths as *G-discretised coordinated paths* (see also Figure 15). It can be shown that solving G-discretised problems (instead of continuous ones) is sufficient to guarantee *probabilistic completeness* of our multi-robot planning scheme, if the simple roadmaps are computed with *PPP* [46].

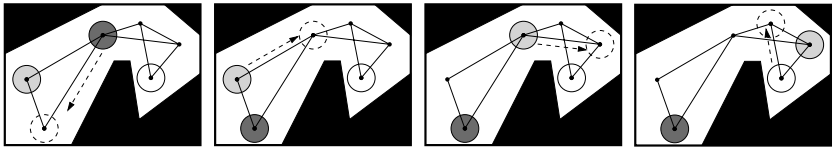


Fig. 15. A G-discretised coordinated path for 3 translating disc-robots.

7.2 The super-graph approach

The question now is, given a simple roadmap $G = (V, E)$ for a robot $\mathcal{A}$, how to compute G-discretised coordinated paths for the composite robot $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ (with $\forall i : \mathcal{A}_i = \mathcal{A}$). For this we introduce the notion of *super-graphs*, that is, roadmaps for the composite robots obtained by combining n simple roadmaps together. We discuss two types of such super-graphs. First, in Section 7.2, we describe a fairly straightforward data-structure, which we refer to as *flat super-graphs*. Its structure is simple, and its construction can be performed in a very time-efficient manner. However, its memory consumption increases dramatically as the number of robots goes up. For reducing this memory consumption (and, through this, increasing the planners power), we generalise this "flat" structure to a multi-level one, in Section 7.2. This results in what we refer to as *multi-level super-graphs*.

Using flat super-graphs In a *flat super-graph* $\mathcal{F}_G^n$, each node corresponds to a feasible placement of the n simple robots at nodes of G, and each edge corresponds to a motion of exactly one simple robot along a non-blocked local path of G. So $((x_1, \ldots, x_n), (y_1, \ldots, y_n))$, with all $x_i \in V$ and all $y_i \in V$, is an edge in $\mathcal{F}_G^n$ if and only if (1) $x_i \neq y_i$ for exactly one i and (2) (x_i, y_i) is an edge in E not blocked by any x_j with $j \neq i$. $\mathcal{F}_G^n$ can be regarded as the Cartesian product of n simple roadmaps. See Figure 16 for an example of a simple roadmap with a corresponding flat super-graph. Any path in the G-induced super-graph describes a G-discretised coordinated path (for the composite robot), and vice-versa. Hence, the problem of finding G-discretised coordinated paths for our composite robot reduces to graph searches in $\mathcal{F}_G^n$. A drawback of flat super-graphs is their size, which is exponential in n (the number of robots). For a formal definition of the flat super-graph method we refer to [46].

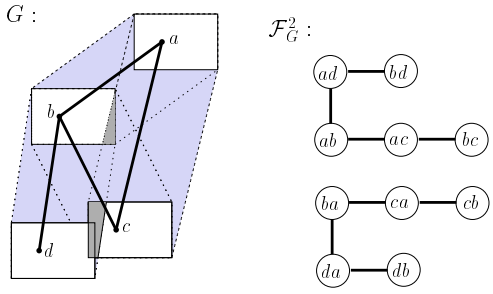


Fig. 16. At the left we see a simple roadmap G for the shown rectangular robot $\mathcal{A}$ (shown in white, placed at the graph nodes). We assume here that $\mathcal{A}$ is a translational robot, and the areas swept by the local paths corresponding to the edges of G are indicated in light grey. At the right, we see the flat super-graph $\mathcal{F}^2_G$, induced by G for 2 robots. It consists of two separate connected components.

Using multi-level super-graphs The *multi-level super-graph method* aims at size reduction of the multi-robot data-structure, by combining multiple node-tuples into single super-nodes. While a node in a flat super-graph corresponds to a statement that each robot $\mathcal{A}_i$ is located at some particular *node* of G, a node in a multi-level super-graph corresponds to a statement that each robot $\mathcal{A}_i$ is located in some *subgraph* of G. But only subgraphs that do not *interfere* with each other are combined. We say that a subgraph A interferes with a subgraph B if a node of A blocks a local path in B, or vice versa. Due to space limitations, we cannot go into much formal details regarding multi-level super-graphs. Here we will just describe the main points. The two main questions are how to obtain the subgraphs, and how to build a super-graph from these in a proper way.

For obtaining suitable subgraphs, we compute a recursive subdivision of the simple roadmap $G = (V, E)$, a so-called *G-subdivision tree* $\mathcal{T}$. Its nodes consist of connected subgraphs of G, induced by certain subsets of V. The root of $\mathcal{T}$ is the whole graph G. The children $(\tilde{V}_1, \tilde{E}_1), \ldots, (\tilde{V}_1, \tilde{E}_1)$ of each internal node $(\tilde{V}, \tilde{E})$ are chosen such that $\tilde{V} = \bigcup_{1 \le i \le k} \tilde{V}_i$ and $\bigcap_{1 \le i \le k} \tilde{V}_i = \emptyset$. Furthermore, all leafs, consisting of one node and no edges, lie at the same level of the tree $\mathcal{T}$. This of course in no way defines a unique G-subdivision tree. We just give a brief sketch of the algorithm that we use for their construction. After the root r (=G) has been created, a number of its nodes are selected heuristically, and subgraphs are grown around these "local roots", until all nodes of r lie in some subgraph. These subgraphs form the children of r, and the procedure is applied

recursively to each of these. The recursion stops at subgraphs consisting of just one node, and care is taking to build a perfectly balanced tree.

For n robots, a simple roadmap $G = (V, E)$ together with a G-subdivision tree $\mathcal{T}$ uniquely defines a *multi-level super-graph* $\mathcal{M}^n_{G\mathcal{T}}$. A n-tuple $(X_1, \ldots, X_n)$ of equal-level nodes of $\mathcal{T}$ is a *node* of $\mathcal{M}^n_{G\mathcal{T}}$ if and only if all subgraphs X_i in the tuple are mutually non-interfering. We define the edges in $\mathcal{M}^n_{G,\mathcal{T}}$ in terms of the flat super-graph $\mathcal{F}^n_G$ induced by G. A pair of super-nodes $((X_1, \ldots, X_n), (Y_1, \ldots, Y_n))$ forms an *edge* E in $\mathcal{M}^n_{G,\mathcal{T}}$ if and only if there exists an edge $e = ((x_1, \ldots, x_n), (y_1, \ldots, y_n))$ in $\mathcal{F}^n_G$ with, for all $i \in \{1, \ldots, n\}$, x_i being a node of X_i and y_i being a node of Y_i. We refer to e as the *underlying flat edge* of E. Also, for the $i \in \{1, \ldots, n\}$ with $x_i \neq y_i$, we refer to the simple robot $\mathcal{A}_i$ as the *active robot* of E (and to the others as the *passive* robots).

We want to stress here that the flat super-graph $\mathcal{F}^n_G$, which can be enormous for $n > 3$ (that is, more than 3 robots), is only used for definition purposes. For the actual construction of our multi-level graph $\mathcal{M}^n_{G\mathcal{T}}$ we fortunately need not to compute $\mathcal{F}^n_G$.

Simulation results show that the size of multi-level super-graphs is considerably smaller than that of equivalent flat super-graphs. Further size-reduction can be achieved by using what we refer to as *sieved* multi-level super-graphs. From experiments we have observed that the connectivity of the free configurations space of the composite robot is typically captured by only a quite small portion of $\mathcal{M}^n_{G\mathcal{T}}$, namely by that portion constructed from the relatively large subgraphs in $\mathcal{T}$. For this reason, we construct $\mathcal{M}^n_{G\mathcal{T}}$ incrementally. We sort the subgraphs in $\mathcal{T}$ by size, and pick them in reversed order of size. For each such picked subgraph we extend the super-graph $\mathcal{M}^n_{G,\mathcal{T}}$ accordingly. By keeping track of the connected components in $\mathcal{M}^n_{G\mathcal{T}}$ we can determine the moment at which the free space connectivity has been captured, and at this point the super-graph construction is stopped.

7.3 Retrieving the coordinated paths

Paths from multi-level super-graphs do not directly describe coordinated paths (as opposed to paths from flat super-graphs). For retrieving a coordinated path from a multi-level super-graph $\mathcal{M}^n_{G\mathcal{T}}$, first the start and goal configurations must be connected by coordinated paths to nodes X and Y of $\mathcal{M}^n_{G,\mathcal{T}}$. Such *retraction paths* can be computed by probabilistic motions of the simple robots. Then, a path $P_{\mathcal{M}}$, connecting X and Y in $\mathcal{M}^n_{G\mathcal{T}}$, must be found, and transformed to a coordinated path P. For each edge E in $P_{\mathcal{M}}$, we do the following: First, we identify the underlying simple edge $e = (a, b)$, and, within its subgraph, we move the active robot to a. Then, we move all passive robots to nodes within their subgraphs that do not block e. And finally we move the

active robot to b (again within its subgraph), over the local path corresponding to e. Applied to all the consecutive edges of $P_{\mathcal{M}}$, this yields a coordinated path that, after concatenation with the two retraction paths, solves the given multi-robot path planning problem.

It follows rather easily from the definition of multi-level super-graphs that the described transformation is always possible.

7.4 Application to car-like robots

We have applied both the flat super-graph method as well as the multi-level super-graph method to *car-like* robots. We have implemented the planners in C++, and tested them on a number of realistic problems, involving up to 5 car-like robots moving in the same environment. Below, we give simulation results from experiments performed with the multi-level super-graph method, for two different environments. The planner was again run on a Silicon Graphics Indigo^2 workstation with an R4400 processor running at 150 MHZ, rated with 96.5 SPECfp92 and 90.4 SPECint92 on the SPECMARKS benchmark.

For both scenes we have first constructed a simple roadmap with *PPP*. The sizes and densities of the two constructed simple roadmaps are sufficient to allow for the existence of G-discretised solutions to most non-pathetic problems in the scenes. Then, we have constructed the multi-level super-graphs incrementally by picking the subgraphs from the G-subdivision tree in order of decreasing size, as described in Section 7.2. We stopped the construction at the point were the multi-level super-graphs consisted of just one major component.

We report the sizes of the resulting super-graphs $\mathcal{M}^n_{G\mathcal{T}} = (V_{\mathcal{M}}, E_{\mathcal{M}})$ and the time required for their construction. Also we give indications of the times required for retrieving and smoothing coordinated paths from the resulting super-graphs. Smoothing is quite essential for obtaining practical solutions, because the coordinated paths retrieved directly are typically very long and "ugly". We use heuristic algorithms for reducing the lengths of the coordinated paths (For details, see [46]).

The left half of Figure 17 shows the first scene, together with the simple roadmap G, consisting of 132 nodes and 274 edges, constructed in about 14 seconds. In the table below we shown the sizes and the construction times of the induced multi-level super-graphs, for 3, 4, and 5 robots. Retrieving and smoothing coordinated paths required, roughly, something between 10 seconds (for 3 robots) and 20 seconds (for 5 robots). See Figure 18 for a path retrieved from the supergraph for 5 robots.

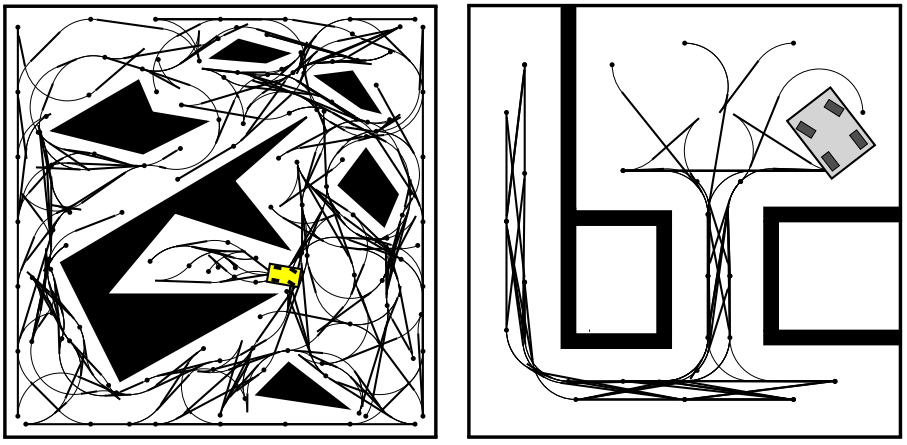


Fig. 17. Two scenes for the multi-robot path planner. Both scenes are shown together with a simple roadmap G for the indicated rectangular car-like robot. Not the edges, but the corresponding local paths are shown.

n	$\|V_{\mathcal{M}}\|$	$\|E_{\mathcal{M}}\|$	Time
3	408	2532	18.5
4	2256	15216	18.8
5	7080	33120	23.3

The right half of Figure 17 shows the second scene on which we test the multi-robot planner. In the table below, the sizes and the construction times of the induced multi-level super-graphs, for 3 and 4 robots, are given. Here, retrieving and smoothing coordinated paths required was easier. Roughly, it took about 6 seconds for 3 robots and 8 seconds for 4 robots. See Figure 19 for a path retrieved from the supergraph for 4 robots.

n	$\|V_{\mathcal{M}}\|$	$\|E_{\mathcal{M}}\|$	Time
3	3018	15630	1.2
4	29712	152016	8.1

We see that the data-structures in the second scene are considerably larger than those required for the first, although the first scene seems to be more complex. The cause for this must be that the compact structure of the free space in the second scene as well as the relatively large size of the robot cause more subgraphs to interfere. Hence, in the second scene, subdivision into smaller subgraphs is required.

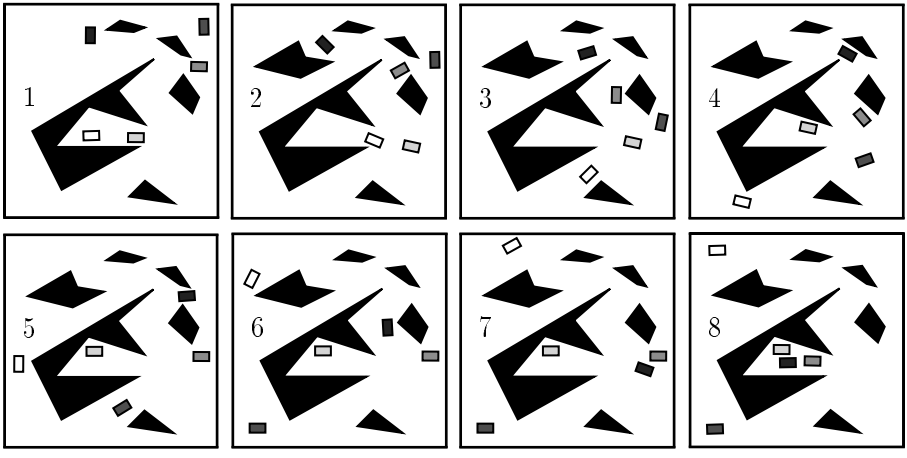


Fig. 18. Snapshots of a coordinated path in the first scene for 5 robots, retrieved from the multi-level super-graph.

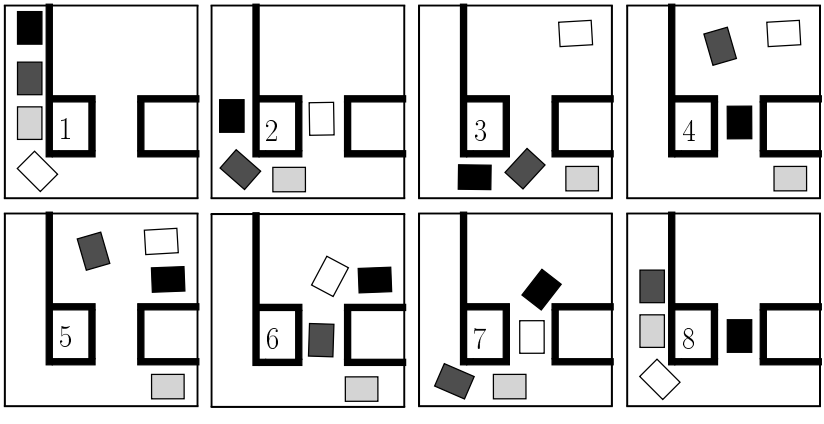


Fig. 19. Snapshots of a coordinated path in the second scene for 4 robots, retrieved from the multi-level super-graph.

7.5 Discussion of the super-graph approach

The presented multi-robot path planning approach seems to be quite flexible, as well as time and memory efficient. The power of the presented approach lies in the fact that only self-collision avoidance is dealt with for the composite robot, while all other (holonomic and nonholonomic) constraints are solved in the C-spaces of the simple robots.

There remain many possibilities for future improvements. For example, smarter ways of building the G-subdivision trees probably exist. For many applications, it even seems sensible to use characteristics of the workspace geometry for determining the subgraphs in the G-subdivision tree. Also, techniques for analysing the expected running times need to be developed.

We have seen that for up to 5 independent robots the method proves practical. However, in many applications one has to deal with much larger fleets of mobile robots. Due to the enormous complexity of such systems, only decoupled planners can be used here. Decoupled planners however can fall into deadlocks. Centralised planners could be integrated into existing large scale decoupled planners for resolving deadlock situations in specific (local) workspace areas where these could arise. For example, if $\mathcal{R}$ is such an area, the global decoupled planner could enforce a simple rule stating that, at any time instant, no more than say 4 robots are allowed to be present in $\mathcal{R}$. Path planning within $\mathcal{R}$ can then be done by a centralised planner, like for example the planner presented in this section.

8 Conclusions

In this chapter an overview has been given on a general probabilistic scheme *PPP* for robot path planning. It consists of two phases. In the roadmap construction phase a probabilistic roadmap is incrementally constructed, and can subsequently, in the query phase, be used for solving individual path planning problems in the given robot environment. So, unlike other probabilistically complete methods, it is a learning approach. Experiments with applications of *PPP* to a wide variety of path planning problems show that the method is very powerful and fast. Another strong point of *PPP* is its flexibility. In order to apply it to some particular robot type, it suffices to define (and implement) a robot specific local planner and some (induced) metric. The performance of the resulting path planner can, if desired, be further improved by tailoring particular components of the algorithm to some specific robot type.

Important is that probabilistic completeness, for holonomic as well as nonholonomic robots, can be obtained by the use of local planners that respect certain general topological properties. Furthermore, there exist some recent results that, under certain geometric assumptions on the free C-space, link the

expected running time and failure probability of the planner to the size of the roadmap and characteristics of paths solving the particular problem. For example, under one such assumption, it can been shown that the expected size of a probabilistic roadmap required for solving a problem grows only logarithmically in the complexity of the problem.

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. *Le Fil d'Ariadne: Une Méthode de Planification Générale. Application á la Planification Automatique de Trajectoires.* PhD thesis, l'Institut National Polytechnique de Grenoble, Grenoble, France, September 1994.
2. R. Alami, F. Robert, F. Ingrand, and S. Suzuki. Multi-robot cooperation through incremental plan-merging. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 2573–2578, Nagoya, Japan, 1995.
3. J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. To appear in *Intern. Journal of Rob. Research.*
4. J. Barraquand and J.-C. Latombe. A Monte-Carlo algorithm for path planning with many degrees of freedom. In *Proc. IEEE Intern. Conf. on Robotics and Automation*, pages 1712–1717, Cincinnati, OH, USA, 1990.
5. J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Internat. Journal Robotics Research.*, 10(6):628–649, 1991.
6. J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.
7. P. Bessière, J.M. Ahuactzin, E.-G. Talbi, and E. Mazer. The Ariadne's clew algorithm: Global planning with local methods. In *Proc. The First Workshop on the Algorithmic Foundations of Robotics*, pages 39–47. A. K. Peters, Boston, MA, 1995.
8. S.J. Buckley. Fast motion planning for multiple moving robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 322–326, Scottsdale, Arizona, USA, 1989.
9. J.F. Canny. *The Complexity of Robot Path Planning.* MIT Press, Cambridge, USA, 1988.
10. M. Erdmann and T. Lozano-Pérez. On multiple moving objects. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1419–1424, San Francisco, CA, USA, 1986.
11. P. Ferbach. A method of progressive constraints for nonholonomic motion planning. Technical report, Electricité de France. SDM Dept., Chatou, France, September 1995.

12. C. Fernandes, L. Gurvits, and Z.X. Li. Optimal nonholonomic motion planning for a falling cat. In Z. Li and J.F. Canny, editors, *Nonholonomic Motion Planning*, Boston, USA, 1993. Kluwer Academic Publishers.
13. J. Hopcroft, J.T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the warehouseman's problem. *International Journal of Robotics Research*, 3(4):76–88, 1984.
14. 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*, pages 3318–3323, San Diego, USA, 1994.
15. Y. Hwang and N. Ahuja. Gross motion planning—a survey. *ACM Comput. Surv.*, 24(3):219–291, 1992.
16. Y.K. Hwang and P.C. Chen. A heuristic and complete planner for the classical mover's problem. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 729–736, Nagoya, Japan, 1995.
17. B. Langlois J. Barraquand and J.-C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Trans. on Syst., Man., and Cybern.*, 22(2):224–241, 1992.
18. P. Jacobs, J.-P. Laumond, and M. Taïx. A complete iterative motion planner for a car-like robot. *Journees Geometrie Algorithmique*, 1990.
19. L. Kavraki. *Random networks in configuration space for fast path planning.* Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, California, USA, January 1995.
20. L. Kavraki, M.N. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. In *IEEE International Conference on Robotics and Automation*, pages 3020–3026, Minneapolis, MN, USA, 1996.
21. L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 2138–2145, San Diego, USA, 1994.
22. L. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. In *Proc. 27th Annual ACM Symp. on Theory of Computing (STOC)*, pages 353–362, Las Vegas, NV, USA, 1995.
23. L. Kavraki, P. Švestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Trans. Robot. Autom.*, 12:566–580, 1996.
24. F. Lamiraux and J.-P. Laumond. On the expected complexity of random path planning. In *Proc. IEEE Intern. Conf. on Robotics and Automation*, pages 3014–3019, Mineapolis, USA, 1996.
25. J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, USA, 1991.
26. J.-P. Laumond, P.E. Jacobs, M. Taïx, and R.M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Trans. Robot. Autom.*, 10(5), October 1994.
27. J.-P. Laumond, S. Sekhavat, and M. Vaisset. Collision-free motion planning for a nonholonomic mobile robot with trailers. In *4th IFAC Symp. on Robot Control*, pages 171–177, Capri, Italy, September 1994.
28. 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.

29. Y.H. Liu, S. Kuroda, T. Naniwa, H. Noborio, and S. Arimoto. A practical algorithm for planning collision-free coordinated motion of multiple mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1427–1432, Scottsdale, Arizona, USA, 1989.
30. P.A. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robotic manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 484–489, Scottsdale, Arizona, USA, 1989.
31. M.H. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, October 1992.
32. M.H. 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.
33. P. Pignon. *Structuration de l'Espace pour une Planification Hiérarchisée des Trajectoires de Robots Mobiles.* Ph.D. thesis, LAAS-CNRS and Université Paul Sabatier de Toulouse, Toulouse, France, 1993. Report LAAS No. 93395 (in French).
34. J.A. Reeds and R.A. Shepp. Optimal paths for a car that goes both forward and backward. *Pacific Journal of Mathematics*, 145(2):367–393, 1991.
35. J.H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Proc. 25th IEEE Symp. on Foundations of Computer Science*, pages 144–154, 1985.
36. J.T. Schwartz and M. Sharir. Efficient motion planning algorithms in environments of bounded local complexity. Report 164, Dept. Comput. Sci., Courant Inst. Math. Sci., New York Univ., New York, NY, 1985.
37. J.T. Schwartz and M. Sharir. On the 'piano movers' problem: III. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal obstacles. *International Journal of Robotics Research*, 2(3):46–75, 1983.
38. S. Sekhavat and J.-P. Laumond. Topological property of trajectories computed from sinusoidal inputs for nonholonomic chained form systems. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 3383–3388, April 1996.
39. S. Sekhavat, P. Švestka, J.-P. Laumond, and M.H. Overmars. Probabilistic path planning for tractor-trailer robots. Technical Report 96007, LAAS-CNRS, Toulouse, France, 1995.
40. S. Sekhavat, P. Švestka, J.-P. Laumond, and M.H. Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystems. To appear in *Intern. Journal of Rob Research.*
41. M. Sharir and S. Sifrony. Coordinated motion planning for two independent robots. In *Proceedings of the Fourth ACM Symposium on Computational Geometry*, 1988.
42. P. Souères and J.-P. Laumond. Shortest paths synthesis for a car-like robot. *IEEE Trans. Automatic Control*, 41:672–688, 1996.
43. H.J. Sussmann. Lie brackets, real analyticity and geometric control. In R.W. Brockett, R.S. Millman, and H.J. Sussmann, editors, *Differential Geometric Control Theory.* Birkhauser, 1983.

44. H.J. Sussmann. A general theorem on local controllability. *SIAM Journal on Control and Optimization*, 25(1):158–194, January 1987.
45. 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.
46. P. Švestka and M.H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 1631–1636, Nagoya, Japan, 1995.
47. P. Švestka and M.H. Overmars. Motion planning for car-like robots using a probabilistic learning approach. *Intern. Journal of Rob Research*, 16:119–143, 1995.
48. P. Švestka and M.H. Overmars. Multi-robot path planning with super-graphs. In *Proc. CESA'96 IMACS Multiconference*, Lille, France, July 1996.
49. P. Švestka and J. Vleugels. Exact motion planning for tractor-trailer robots. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 2445–2450, Nagoya, Japan, 1995.
50. D. Tilbury, R. Murray, and S. Sastry. Trajectory generation for the n-trailer problem using goursat normal form. In *Proc. IEEE Internat. Conf. on Decision and Control*, San Antonio, Texas, 1993.
51. P. Tournassoud. A strategy for obstacle avoidance and its application to multi-robot systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1224–1229, San Francisco, CA, USA, 1986.
52. S.M. La Valle and S.A. Hutchinson. Multiple-robot motion planning under independent objectives. To appear in *IEEE Trans. Robot. Autom.*.
53. F. van der Stappen. *Motion Planning amidst Fat Obstacles.* Ph.D. thesis, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, October 1994.
54. F. van der Stappen, D. Halperin, and M.H. Overmars. The complexity of the free space for a robot moving amidst fat obstacles. *Comput. Geom. Theory Appl.*, 3:353–373, 1993.