On Delaying Collision Checking in PRM Planning – Application to Multi-Robot Coordination

Gildardo Sánchez[(1)] and Jean-Claude Latombe[(2)]

(1) ITESM, Campus Cuernavaca, Cuernavaca, México
(2) Computer Science Department, Stanford University, Stanford, CA, USA
Emails: gsanchez@campus.gda.itesm.mx, latombe@cs.stanford.edu

Abstract: This paper describes the foundations and algorithms of a new probabilistic roadmap (PRM) planner that is: *single-query* – instead of pre-computing a roadmap covering the entire free space, it uses the two input query configurations to explore as little space as possible; *bi-directional* – it explores the robot's free space by building a roadmap made of two trees rooted at the query configurations; and *lazy in checking collisions* – it delays collision tests along the edges of the roadmap until they are absolutely needed. Several observations motivated this strategy: (1) PRM planners spend a large fraction of their time testing connections for collision; (2) most connections in a roadmap are not on the final path; (3) the collision test for a connection is most expensive when there is no collision; and (4) any short connection between two collision-free configurations has high prior probability of being collision-free. The strengths of single-query and bi-directional sampling techniques, and those of delayed collision checking reinforce each other. Experimental results show that this combination reduces planning times by large factors, making it possible to efficiently handle difficult planning problems, such as problems involving multiple robots in geometrically complex environments. This paper specifically describes the application of the planner to multi-robot planning and compares results obtained when the planner uses a centralized planning approach (PRM planning is then performed in the joint configuration space of the robots) and when it is uses a decoupled approach (the PRM planner is invoked several times, first to compute a path of each robot independent of the others, and then to coordinate those paths). On a simulated six-robot welding station combining 36 degrees of freedom, centralized planning has proven to be a much more effective approach than decoupled planning.

1. Introduction

Probabilistic roadmaps (PRM) and related methods are an effective tool to solve path-planning problems with many degrees of fredom (dofs) [Ahu94, ABD+98, BKL+97, BK00, BL91, BOvdS99, HST94, Hsu00, HLM+99, HXCW98, Kav94, KSLO96, LS01, NSL99, Sve97]. They also make it possible to handle various admissibility constraints, such as nonholonomic and kinodynamic constraints [Hsu00, KHLR00, Kuff99, LK99, LK01, SSLO98, SO95], maintenance of stability [Cas01, KNK+01], avoidance of moving obstacles [Bag96, KHLR00], deformable objects [GHK99, HKW98, LamK99], closed-loop kinematic chains [HA01, LYK99], and motion in contact [JX01]. PRM and related planners have not only been used for basic trajectory generation. They have also been applied to (dis-)assembly and manipulation planning [CL95, KL94, NK00, SLL01, SRA01], finding the placement of a robot arm [HLS99], planning the motions of air-cushion robots dodging moving obstacles [Kin01], planning aggressive maneuvers of autonomous helicopters [FDF00], locomotion and manipulation planning for humanoid robots [KNK+01], reconfiguration planning for modular self-reconfigurable robots [Cas01], solving inverse kinematic problems [AG99], multi-robot coordination [AdB+99], multi-goal planning [SR00], graphic animation of digital characters [CLS00, KKKL94, Kuff99], surgical planning [TAL99], and molecular motion prediction [ABG+02, SLB99, SA01].

A PRM planner samples the configuration space at random and retains the collision-free points as *milestones*. It connects pairs of milestones by simple paths and retains the collision-free ones as *local paths*. The milestones and local paths form the *probabilistic roadmap*. The motivation is that, while it is often impractical to compute an explicit geometric representation of the collision-free subset (the *free space*) of a configuration space, existing collision-detection algorithms can efficiently check whether a given configuration or local path is

collision-free [BKL+97]. Under broad assumptions, the probability that an adequately designed PRM planner finds a collision-free path, if one exists, goes to 1 quickly in the number of milestones [HLM97, Hsu00, KKL98, LK02, Sve97]. This property is called *probabilistic completeness*.

Not surprisingly, PRM planners spend most of their time performing collision checks (often more than 99%). Several approaches reviewed in Section 2.1 have been proposed to reduce the total cost of collision checking. They consist of designing faster collision-checking algorithms, as well as smarter sampling and connection strategies. In this paper, we propose a connection strategy that (1) delays collision tests until they are absolutely needed to check that a candidate path is a solution, and (2) schedules these tests along the path in order to reveal a collision as early as possible, if there is one. This strategy is related to those introduced in [BK00, NK00], but it is combined here with a single-query bi-directional sampling strategy similar to the one in [HLM97, Hsu00]. We name the resulting planner SBL – for Single-query, Bi-directional, Lazy collision-checking planner.

More precisely, SBL concurrently builds and searches a network of milestones made of two trees rooted at the input query configurations, hence focusing its attention to the subset of the free space that is reachable from these configurations. It also locally adjusts the sampling resolution, taking larger steps in regions of the free space for which there is evidence that they are wide-open, and smaller steps in regions that appear to be cluttered with obstacles. Like in [BK00] it does not immediately test connections between milestones for collision. When a sequence of milestones joining the two query configurations is created, then (and only then) the connections between milestones along this path are tested. In a way related to [NK00] this test is performed at successive points carefully ordered according to their likelihood of revealing a collision. Hence, no time is wasted testing connections that are not on a candidate path, and relatively little time is spent checking connections that are not collision-free. On a 1-GHz Pentium III processor, SBL reliably solves problems for 6-dof robots in environments whose geometry is described by several dozen thousand triangles, in times ranging from a small fraction of a second to a few seconds. Comparison with a single-query bi-directional planner using a traditional collision-checking strategy shows that the lazy strategy cuts the number of collision tests (and the total planning time) by a large factor. In our experiments, this factor ranged between 4 and 40.

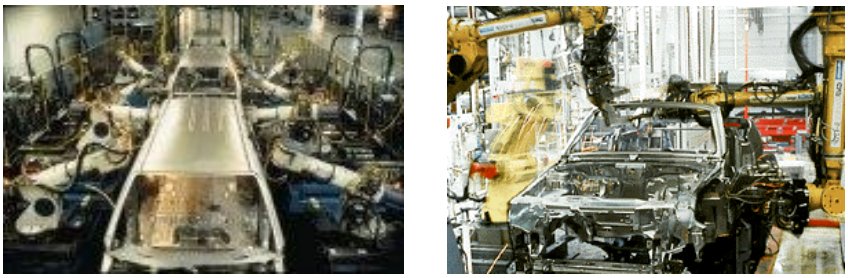


Figure 1: Spot-welding stations in automotive body shops

This increased efficiency makes it possible to use SBL on complex tasks. One particularly interesting task is the programming of multi-robot welding stations in automotive body shops (Figure 1). A typical such station include 4 to 10 robots, each sharing a portion of its workspace with 1 to 4 other robots. The geometric model of a station (robots and car bodies) may consist of several 100,000 triangles. The task of manually programming a spot-welding station is long, fastidious, and prone to errors. Late adjustments in the positions of weld points (such as those suggested by data collected during crash-tests) may cause manufacturing interruptions, as some robot trajectories must be modified. It is then highly desirable to generate the new trajectories as quickly as possible. Using this example as a source of inspiration, we have extended SBL to solve multi-robot problems. There exist two traditional approaches to multi-robot motion planning: centralized and decoupled planning. Centralized planning has been seldom used in the past because it requires searching a high-dimensional joint configuration space. Instead, decoupled planning breaks the planning problem into smaller problems, but the approach is inherently incomplete. SBL can be used to implement either approach. We experimented with it on a six-robot welding station combining 36 dofs. When performing centralized planning, SBL has been fully reliable and reasonably fast. When performing decoupled planning, it was marginally faster (when it succeeded in finding a solution), but much less reliable (due to the incompleteness of the approach, it sometimes failed to

find a solution). This observation is important as it invalidates the prevailing assumption that the loss of completeness in performing decoupled planning is not very significant in practice and worth the computational gain. Our results indicate instead that centralized planning may be a much more desirable approach, and that the existence of efficient PRM planners such as SBL makes this approach technically feasible.

This paper is organized into two main sections. Section 2 first describes the basic planner. Subsection 2.1 relates the proposed sampling and connection strategies to prior work in PRM planning. Subsection 2.2 establishes the foundations of the lazy collision checking strategy. Subsection 2.3 describes SBL's algorithms. Subsection 2.4 presents experimental results on a single robot. Section 3 then describes the application of SBL to multi-robot planning. It also relates our work to previous research on multi-robot planning.

This paper extends the descriptions in [SL01, SL02], while a more extensive presentation can be found in [San02]. The code of SBL can be downloaded from http://robotics.stanford.edu/~latombe/projects/ .

Notations: Throughout this paper, C denotes the robot's configuration space and $F \subset C$ its free space. We normalize the range of values of each dof to be [0,1], and we represent C as $[0,1]^n$, where n is the number of dofs of the robot. We define a metric d over C. The metric used in SBL is the L_∞ metric, but others would work as well. For any $q \in C$, the *neighborhood* of q of radius r is $B(q,r) = \{q' \in C \mid d(q,q') < r\}$.

2. Basic Planner

2.1. Relation to previous work

The goal of the techniques embedded in SBL – lazy collision checking combined with single-query bi-directional sampling – is to speed up PRM planning by spending less time in unnecessary collision tests. Here, we review existing techniques related to this same goal. We organize them into three groups: (1) collision-checking techniques, which directly affect the performance of PRM planning; (2) milestone sampling strategies, since smart sampling strategies may make it possible to answer planning queries correctly with smaller roadmaps; and (3) connection strategies, as a large fraction of the collision checks are done to test connections between milestones.

2.1.1. Collision checking

PRM planners heavily rely on fast collision-checking algorithms to test both sampled milestones and connections between milestones. Collision tests must scale up to complex geometric environments. A common practice is to represent objects by their triangulated surfaces. In practical problems, models may contain several 100,000 triangles, or more. Recently, a number of efficient algorithms have been proposed [Lin00]:

(1) *Feature-based methods* [Bar90, LC91, Mir98] exploit temporal and spatial coherence in the dynamic geometric model to maintain the pair of closest features (vertices, edges, faces) between the robot and the static obstacles. This pair is first computed at the initial configuration of the robot along a given path τ. It is then updated along this path by exploiting the facts that (1) it does not change continuously, but only at discrete configurations of the robot, and (2) each change yields an update that can be computed at relatively small cost.

(2) *Hierarchical decomposition methods* pre-compute a hierarchy of bounding volumes for each object (robot link, obstacle) [CLMP95, Fav89, GLM96, KHM98, KPLM98, Qui94]. During a collision test, large subsets of the objects whose bounding volumes do not intersect are quickly discarded. Several bounding volumes have been proposed: spheres [Hub95, PG95, PSI92, Qui94], axis-aligned bounding boxes (AABBs) [CLMP95], oriented bounding boxes (OBBs) [GLM96], discrete-orientation polytopes (DOPs) [KHM98], and sphere shells [KPLM98]. Some volumes (e.g., OBBs and DOPs) yield smaller hierarchies, but intersection tests on such volumes are slightly more costly.

Although each approach has distinct advantages and drawbacks, hierarchical decomposition methods have been more widely used so far. Feature-based algorithms have two major drawbacks as far as PRM planning is concerned: (1) they assume that collision tests are made while the robot is continuously displaced along a certain path, hence do not apply to check configurations picked at random; (2) they work well only when the objects can be described by small collections of convex pieces, which is rarely the case in robotics.

In PRM planners, the hierarchical decomposition approach has already proven to scale up well to environments where object surfaces are described by several 100,000 triangles [HLM97]. Our own experiments on several types of objects show that the PQP checker [GLM96] can test two non-convex objects whose triangulated surfaces are described by 500,000 triangles each in times ranging between .0001 to .002 seconds on a 1-GHz Pentium III processor. (The pre-processing time is on the order of 60 seconds.) Tests are particularly fast when the objects are well separated and when they actually intersect. In practice, the time required by a collision test tends to grow logarithmically in the number of triangles (although it is linear in the worst case).

A simple way to test a connection between two milestones for collision is to discretize the connection into a sequence of segments shorter than some given ε, and to return that the connection is collision-free if the endpoints of all the segments are collision-free [ABD+00, KSLO96]. This procedure is easily implemented as recursive bisection, but it is not a fully rigorous test. It may miss collisions if ε is set too large, while being un-necessarily slow if ε is set too small.

Instead, one can exploit the ability of some checkers, such as Quinlan's [Qui94] and PQP [GLM96], to compute the Euclidean distance between two objects. Consider a straight-line segment in C between two configurations q to q'. During a motion of the robot along this segment, every point of the robot traces a distinct curve in the workspace. Denote by $L(q,q')$ the Euclidean length of the longest such curve over all robot points. For any given robot, there exists a constant $\alpha > 0$ such that $L(q,q') \leq \alpha \times d(q,q')$, where d is the metric over C. As a simple example, let the robot be a rigid object moving in translation and rotation in a two-dimensional workspace. Let (x,y,θ) be its configuration, where (x,y) are the coordinates of a robot's "reference" point P in the workspace coordinate system and θ is the angle between an axis of this coordinate system and a "reference" vector on the robot. Let the distance $d(q,q')$ between $q = (x,y,\theta)$, and $q' = (x',y',\theta')$ be the weighted L_∞ metric $d(q,q') = \max\{|x'-x|, |y'-y|, R\times|\theta'-\theta|\}$, where R is the maximal distance between P and the robot's contour. When the robot moves along the segment joining q to q', none of its points moves along a curve of Euclidean length greater than $|x'-x| + |y'-y| + r\times|\theta'-\theta| \leq 3 \times d(q,q')$, yielding $\alpha = 3$. Other constants would be established for different metrics in C and other robots. If there exists a series of r points $q_1,\ldots,q_r$ on a robot path τ, where q_1 and q_r are the two endpoints of τ, such that $\alpha \times d(q_i,q_{i+1}) \leq \max\{D(q_i),D(q_{i+1})\}$ for every $i = 0,\ldots,r\text{-}1$, then τ is guaranteed to be collision-free [BKL+97, Hsu00]. So, checking the segment between q to q' can be done as follows. Compute $D(q)$ and D(q'). If any of these two distances is 0, then return that the segment collides. Otherwise, if $\alpha \times d(q,q') \leq \max\{D(q),D(q')\}$ then return that it is collision-free; else break it into two halves and test each half recursively.

Since most of the computational time of a PRM planner is spent testing collision, any improvement in collision checking will translate into a similar speed-up of the planner. But considerable progress has recently been made and further significant progress seems difficult. One approach could be to design a sampling strategy and a checker that work in symbiosis, with the sampling strategy picking each new configuration so that part of the collision-checking work done for previous sampled configurations could be re-used. SBL simply uses an existing checker – PQP [GLM96] – to perform all its collision tests. We have chosen PQP because it is both fast and easy to use (in particular, it needs little parameter tuning).

2.1.2. Milestone sampling strategies

Another way to reduce collision-checking work in PRM planning is to design smarter sampling strategies that avoid generating many milestones in un-interesting areas of the free space F:

(1) *Multi-stage strategies*. The planner in [Kav94] first generates milestones uniformly distributed over F and local paths between these milestones. Next, in a second stage, called the enhancement step, it selects additional milestones around milestones that have few or no connections to other milestones. Indeed, poorly connected milestones tend to lie in narrow areas of F, whose connectivity is more difficult to capture than wide-open areas. The enhancement step yields a greater density of milestones in such narrow areas. Experimental results reported in [Kav94] show that, in general, a roadmap of s milestones is much better connected when roughly $2s/3$ milestones are picked uniformly and $s/3$ milestones are generated during the enhancement step, than when all s milestones are picked uniformly.

(2) *Obstacle-sensitive strategies*. The areas of F whose connectivity is the most difficult to capture by a roadmap are usually close to F's boundary. Instead of discarding configurations sampled in the forbidden region, an obstacle-sensitive strategy uses each such point to cast rays along randomly selected directions. Discrete "walks" along the rays, performed incrementally with a fixed step or using bisection techniques, yield free configurations near the boundary of F [AW96, Ove92]. *Gaussian sampling* is another obstacle-sensitive strategy that retains a sampled configuration as a milestone only if it is collision-free *and* a forbidden configuration has been found in its neighborhood [BOvdS99]. At the limit, this technique produces a Gaussian distribution of milestones whose density peaks near F's boundary and fades away from it. In [HST94] rays are cast from milestones and are reflected at F's boundary to create more milestones near F's boundary.

(3) *Narrow-passage strategies.* A notorious difficulty for PRM planners is finding connections through narrow passages of F [HKL+98, HLM97]. The previous strategies are relevant to this difficulty, but do not address it explicitly. Instead, the strategies in [Bag97, HKL+98] build a first roadmap in a "dilated" free space F' by allowing the robot to slightly penetrate the obstacles. The purpose of the dilatation of F is to widen the narrow passages, hence making it easier to find connections through them. Local re-sampling is then used to generate a collision-free milestone in the neighborhood of every milestone contained in $F' \backslash F$. But this strategy suffers from the fact that techniques for computing penetration distances are much slower than for computing separation distances. Another narrow-passage strategy is to extract the medial-axis or Voronoi diagram of the workspace, and to pick more samples around the branches of the diagram that have small clearance [GHK99, HCK+00, WAS99]. Medial-axis strategies have proven effective when the robot is rather compact and rigid (or with relatively small changes in shape). So far, they have not scaled up well to articulated arms, as it is then difficult to meaningfully relate the medial axis of the workspace to the connectivity of F.

(4) *Single-query strategies*. The above strategies apply well when the roadmap is fully computed before processing path-planning queries. Instead, a PRM planner using a single-query sampling strategy computes a new roadmap for each query [AG99, BL91, HLM97, Kuf99, Hsu00, LK01, VRA01]. It uses its knowledge of the query configurations to only explore restricted subsets of the components of F reachable from these configurations. This is done by growing one tree of milestones from the initial configuration, until a connection is found with the goal configuration (*single-directional* search), or by growing two trees concurrently, respectively rooted at the initial and goal configurations, until a connection is found between them (*bi-directional* search). In both cases, milestones are iteratively added to the roadmap. Each new milestone m' is picked in a neighborhood of an existing milestone m and is connected to m by a local path. In practice, single-query strategies are much faster, per query, than multi-query ones. Often, the cost of pre-computing a roadmap is justified only if several dozens or even hundreds of queries use this roadmap. Bi-directional PRM planners are also usually more efficient than single-directional ones [Hsu00].

(5) *Diffusion strategies*. In order to be effective, tree-growing single-query strategies must carefully avoid over-sampling some areas of F. The roadmap tree(s) they expand must "diffuse" across the components of F reachable from the query configurations. The technique in [HLM97, Hsu00] first selects an existing milestone m at random with probability inverse to the sampling density around m, and next picks a successor of m by sampling a neighborhood of m uniformly at random. The technique in [Kuf99, LK01] picks a configuration q at random in the configuration space, then selects the existing milestone m that is the closest to q, and finally picks a successor of m along the line connecting m to q. In both cases, efficient space-indexing techniques [Aga97] must be used when the number of milestones becomes large, either to update the sampling density around each milestone, or to identify the closest milestone to a given configuration q.

There also exist sampling strategies that are not intended to achieve greater computational efficiency, but to deal with motion constraints other than collision avoidance. For instance, the control-based strategies in [LK99, KHLR00] directly generate local paths that satisfy kino-dynamic constraints.

SBL applies a bi-directional single-query strategy and a diffusion strategy similar to those described in [HLM97, Hsu00]. In addition, it also uses an adaptive technique that results in taking longer steps in wide-open area of *F* than in narrow ones. It is the combination of these sampling techniques with a connection strategy based on delayed collision checking that makes it particularly efficient.

2.1.3. Connection strategies

A third way to reduce collision-checking work is to avoid testing all possible connections between milestones. This is the main approach investigated in this paper. Two types of strategies have been proposed so far:

(1) *Connection reduction strategies*. This type of strategy was introduced in [Kav94] and is applicable during the pre-computation of a roadmap. It consists of testing a reduced set of connections, either by setting a threshold on the length of the tested connections, or by limiting the number of tested connections from each milestone. The algorithm in [Kav94] also maintains an explicit representation of the connected components of a roadmap, while it is being generated, and never tests a connection between two milestones already in the same component. The *visibility-based-roadmap* strategy in [LS01, NSL99] pushes this idea further, by discarding a new milestone *m* if it does not satisfy any of the following two conditions: (1) *m* cannot be connected to any previous milestone (then, *m* is retained in a new component of the roadmap) and (2) *m* can be connected to two previous milestones belonging to separate components of the roadmap (then, the two components are merged into one that includes *m*). Experiments show that this technique yields smaller roadmaps that still cover *F* well. It is not clear, however, whether the relatively high cost of processing each milestone is later recovered by a greater saving.

(2) *Delayed collision-checking strategies*. The underlying idea is to avoid performing collision tests before they are absolutely needed. The *lazy collision-checking* planner of [BK00] generates an initial roadmap by sampling the configuration space uniformly at random. It initially assumes that all points and connections are collision-free; hence, the initial roadmap is the complete graph whose nodes are the sampled configurations. To process a query, it adds the two query configurations to the graph and computes the shortest path between them in the roadmap. Only then it tests the connections along this path for collision. If a collision is found, the node and/or connection where the collision is detected are erased, and a new shortest path is computed and tested; and so on. The *fuzzy-roadmap* planner of [NK00] also generates an initial roadmap, but here the nodes of the roadmaps are tested for collision. The connections are not immediately tested, and a probability is associated with each one indicating the likelihood that it is collision-free. The probability that a path between two nodes of the roadmap is collision-free is computed as the product of the probabilities associated with the connections it contains. The planner computes the path between the query configurations that has the highest probability of being collision-free. Then it tests each connection along this path for collision, starting with the one that has the highest probability to reveal a collision.

We think that delaying collision checks is a promising approach to speed up PRM planners. SBL draws upon the ideas pioneered in [BK00, NK00], but applies them quite differently. It tries to exploit them to a fuller extent, in particular by combining them with a single-query bi-directional sampling strategy. The initial network built by the planners in [BK00, NK00] is reminiscent of a roadmap pre-computed by multi-query planners. One must decide in advance how large it should be. If it is too coarse, it may fail to contain a solution path. But, if it is too dense, time will be wasted checking similar paths that will successively test to be colliding. The choice of the size of the network is made even harder by the fact that in most practical problems the free space occupies a relatively small fraction of the configuration space (i.e., most points picked at random in configuration space eventually test to be colliding). The focus on shortest paths in [BK00] or on paths with highest probability of being collision-free [NK00] may also introduce an inappropriate bias when obstacles force the robot to take long detours. Such difficulties are recognized in [BK00], where a re-sampling step is added to the basic

strategy. This step, which is performed when the basic strategy fails to find a path, adds milestones in regions selected around existing milestones. The planners in [BK00, NK00] are related to "customizable" PRM planners that pre-compute a partial roadmap by performing only approximate validation of the nodes and edges, and complete this roadmap during the query phase [SMA01]. SBL is definitely a single-query planner.

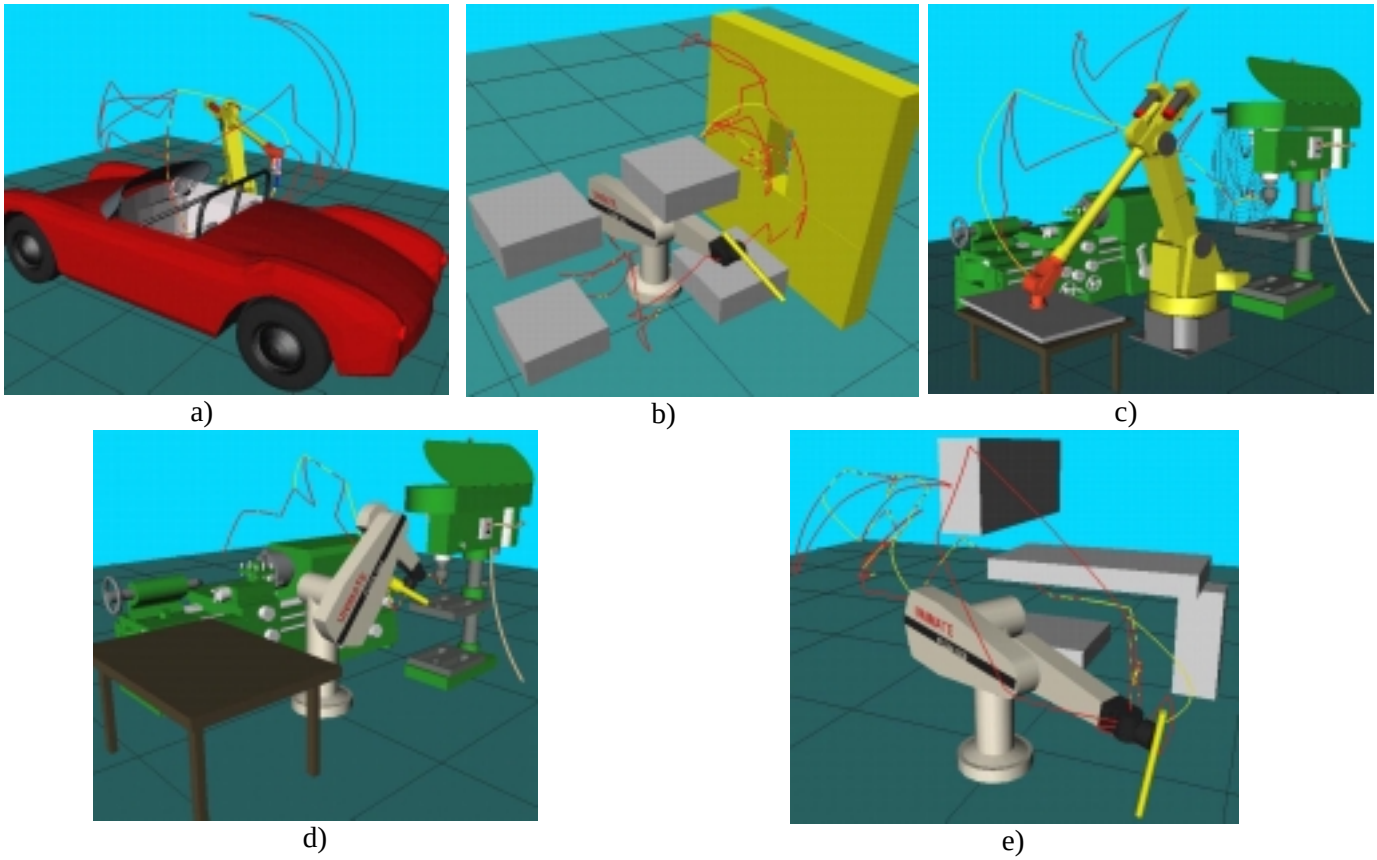


Figure 2: Path planning environments

2.2. Experimental foundations of SBL

The design of SBL was suggested by early experiments that we performed with the single-query PRM planner described in [Hsu00]. To study the impact of collision checking on the running time, we modified the planner's code by removing collision checks for connections between milestones (hence, only the milestones remained tested). The planner was faster by two to three orders of magnitude, but surprisingly a significant fraction of the generated paths were actually collision-free. Figure 2 shows environments in which we made this observation. Every connection created by the planner in [Hsu00] is relatively short – less than 0.3, while the L_∞ diameter of C is 1. Thus, the above observation suggests that if two configurations picked at random are both collision-free (milestones) and close to each other, then the straight-line segment between them has high prior probability of being collision-free. This explanation was confirmed by subsequent experiments presented below.

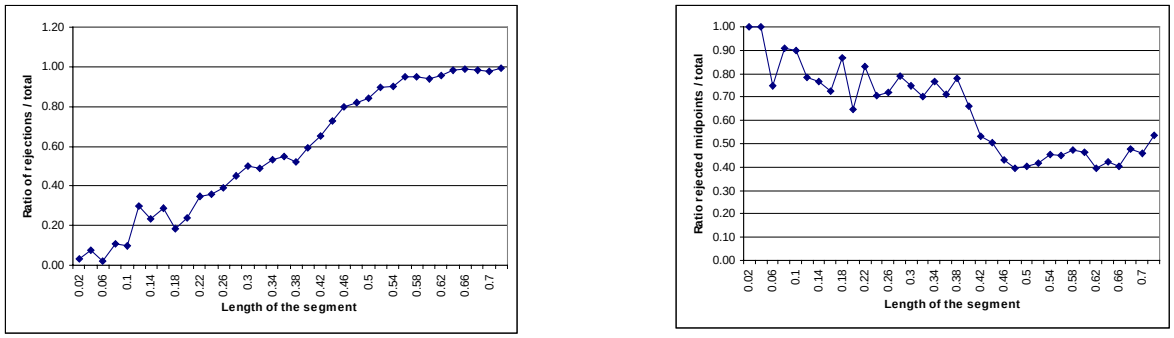
Following this somewhat surprising observation, we performed several additional experiments that eventually resulted in the following key observations, from which our collision-checking strategy is derived:

1. Most local paths in a roadmap are not on the final path. Using the planner of [Hsu00] in the examples of Figure 2, we measured that the ratio of milestones on the final path varies between 0.1 and 0.001.

2. The test of a connection is most expensive when it is actually collision-free. Indeed, the test ends as soon as a collision is detected, but is carried down to the finest resolution when there is no collision.

3. A short connection between two milestones has high prior probability of being collision-free. Thus, testing PRM connections early is likely to be both useless and expensive.

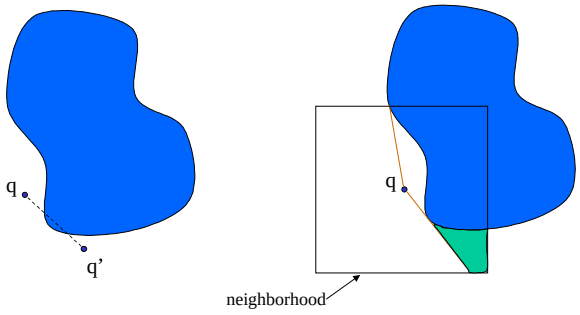
4. If a connection between two milestones is colliding, its midpoint has high probability to be in collision, so that this point should be tested next (a choice that can be applied recursively)



a) b)

Figure 3: Collision ratios along connections

Observations 3 and 4 are made more explicit in Figure 3. We generated 10,000 segments at random with L_∞ lengths uniformly distributed between 0 and 1. This was done by picking 100 collision-free configurations in C uniformly at random and connecting each such configuration q to 100 additional collision-free configurations obtained by randomly sampling neighborhoods of q of different radii. We then tested each of the 10,000 segments for collision. Figure 3a (generated for the environment of Figure 2a) displays the ratio of the number of segments that tested collision-free in each interval. Here, a segment shorter than 0.25 has probability greater than 0.6 of being collision-free. Similar charts were obtained with the other environments. Figure 3b shows, for each interval of Figure 3a, the fraction of colliding segments whose midpoints are colliding. Hence, if a short connection is colliding, its midpoint has high probability to collide.



a) b)

Figure 4: Illustration for the result of Figure 3

There is a simple explanation for the results of Figure 3. Since the robot and most of the obstacles are "thick" in all or most directions, the obstacle regions in C are also thick in most directions (a concept related to that of a "fat obstacle" [SHO93]). Hence, a short colliding segment with collision-free endpoints is necessarily almost tangential to an obstacle region in C, an event that has small probability of happening. Indeed, consider Figure 4, where the dark (blue) region is a thick obstacle region in a fictitious 2-D configuration space. Let q and q' be two configurations picked at random that are both collision-free, close to each other, and such that the straight

segment joining them intersects the obstacle region (Figure 4a). Assume that q has been picked first (necessarily near F's boundary) and that q' is selected next inside a relatively small neighborhood of q. The gray (green) subset of this neighborhood (shown in Figure 4b) is the area in which q' must be selected in order to satisfy the above conditions. In general, this subset is only a small fraction of the neighborhood's collision-free subset, so that the probability of picking q' in it is small. This would not be true if the obstacle regions in C were thin, e.g., if the robot was a point moving in a planar maze of arbitrarily thin walls.

The lazy collision-checking strategy of SBL derives directly from the four observations listed above. These suggest that we should check the sampled configurations for collision before retaining them as milestones, but not the connections to their parents. The test of these connections should be delayed until a sequence of milestones is found that connects the initial and goal configurations. Together, observations 3 and 4 suggest scheduling collision tests so that the midpoint of the longest un-tested segment is always tested next. SBL assumes that obstacle regions in C are thick in most directions, a condition usually satisfied in practical problems.

2.3. Description of SBL

SBL is given two parameters: s – the maximum number of milestones that it is allowed to generate – and ρ – a distance threshold. Two configurations are considered close to one another if their L_∞ distance is less than ρ. In our implementation, ρ is typically set between 0.1 and 0.3. Since SBL does not immediately test connections between milestones for collision, rather than referring to a connection between milestones as a *local path*, we call it a *segment*.

2.3.1. Overall algorithm

Algorithm SBL
1. Install q_{init} and q_{goal} as the roots of T_{init} and T_{goal}, respectively
2. Repeat s times
   2.1. EXPAND
   2.2. $\tau \leftarrow$ CONNECT
   2.3. If $\tau \neq$ *nil* then return τ
3. Return *failure*

SBL builds two milestone trees, T_{init} and T_{goal}, respectively rooted at the configurations q_{init} and q_{goal}. Each loop of Step 2 performs two steps: EXPAND adds a milestone to one of the two trees, while CONNECT tries to connect the two trees. SBL returns *failure* if it has not found a solution path after s iterations at Step 2. In this case, either no solution path exists between q_{init} and q_{goal}, or the planner failed to find one.

2.3.2. Roadmap expansion

Algorithm EXPAND
1. Pick T to be either T_{init}, or T_{goal}, each with probability 1/2
2. Repeat until a new milestone q has been generated
   2.1. Pick a milestone m from T at random, with probability $\pi(m) \sim 1/\eta(m)$
   2.2. For i = 1, 2, … until a new milestone q has been generated
      2.2.1. Pick a configuration q uniformly at random from $B(m,\rho/i)$
      2.2.2. If q is collision-free then install it as a child of m in T

Each expansion of the roadmap consists of adding a milestone to one of the two trees. The algorithm first selects the tree T to expand. The alternation between the two trees (Step 1) prevents one tree from eventually growing much bigger than the other, as we would then lose the advantages of bi-directional search. The number $\eta(m)$ associated with each milestone m in T measures the current density of milestones of T around m. (See implementation details in Subsection 2.3.5.) A milestone m is picked from T with probability proportional to $1/\eta(m)$ and a collision-free configuration q is picked at close distance (less than ρ) from m. This configuration q is the new milestone.

The probability $\pi(m) \sim 1/\eta(m)$ at Step 2.1 was introduced in [HLM97] to avoid over-sampling regions of F. It guarantees that the distribution of milestones eventually diffuses through the component(s) of F reachable from q_{init} and q_{goal}. In [HLM97, Hsu00] this condition is needed to prove that the planner is probabilistically complete with fast convergence. Another diffusion technique is proposed in [Kuf99, LK01] (see Subsection 2.1.2).

Step 2.2 selects a series of milestone candidates, at random, from successively smaller neighborhoods of m, starting with a neighborhood of radius ρ. When a candidate q tests collision-free, it is retained as the new milestone. The segment from m to q is not checked here for collision. On the average, the jump from m to q is greater in wide-open regions of F than in narrow regions. Though not described further in this paper, the impact of this adaptive technique on SBL's efficiency is quite significant.

Since the segment from m to q is not checked for collision, there is a small risk that the tree T "jumps" over an obstacle and expands outside the connected component of F reachable from its root. Our experimental results show that this risk is dwarfed by the computational benefits of postponing collision tests.

2.3.3. Tree connection

Algorithm CONNECT
1. $m \leftarrow$ most recently created milestone
2. $m' \leftarrow$ closest milestone to m in the tree not containing m
3. If $d(m,m') < \rho$ then
   3.1. Connect m and m' by a bridge w
   3.2. $\tau \leftarrow$ path connecting q_{init} and q_{goal}
   3.3. Return TEST-PATH (τ)
4. Return *nil*

Let m now denote the milestone that was just added by EXPAND and m' be the closest milestone to m in the other tree. If m and m' are less than ρ apart, then CONNECT joins them by a segment, called a *bridge*. The bridge creates a path τ joining q_{init} and q_{goal} in the roadmap. The segments along τ, including the bridge, are now tested for collision. TEST-PATH returns *nil* if it detects a collision.

2.3.4. Path testing

Let v be the straight segment connecting two collision-free configurations q and q'. Let COLLISION(v) be the procedure that returns *no* if $\alpha \times d(q,q') \leq \max\{D(q),D(q')\}$ and *maybe* otherwise (see Subsection 2.1.1).

When a segment u is added to the roadmap, COLLISION(u) is invoked; if it returns *no*, then u is marked *safe*. Otherwise, the planner associates with u a set *Unsafe*(u) of unsafe sub-segments, which is initialized to $\{u\}$. TEST-PATH (described below) makes use of the algorithm TEST-SEGMENT(u) that tests the unsafe sub-segments in *Unsafe*(u) by breaking each of them into two halves:

Algorithm TEST-SEGMENT(u)
1. $S \leftarrow$ empty set
2. For every $v \in$ *Unsafe*(u) do
   2.1. Let q_{mid} be the midpoint of v; if $D(q_{\text{mid}}) = 0$ then return *collision*
   2.2. Break v into two halves, v_1 and v_2
   2.3. For i = 1,2 do
      2.3.1. If COLLISION(v_i) = *maybe* then add v_i to S
3. If S is empty then mark u as *safe*; else *Unsafe*(u) $\leftarrow S$

Note that at any one time all sub-segments in *Unsafe*(u) have the same length, which we denote by $\varepsilon(u)$. The smaller $\varepsilon(u)$, the greater the chances that u will actually test collision-free. A side-effect of every call to TEST-SEGMENT(u), if it neither detects a collision, nor marks u as *safe*, is to divide $\varepsilon(u)$ by 2.

Let p be the number of segments in the path τ to be tested by TEST-PATH, and u_1, u_2, …, u_p denote those segments, with u_1 originating at q_{init} and u_p ending at q_{goal}. TEST-PATH(τ) maintains a priority queue U of all the segments in $\{u_1, u_2, \ldots, u_p\}$ that are not marked *safe*. U is sorted in decreasing order of $\varepsilon(u_i)$. Intuitively, the first segment in U is the "most unsafe" segment, i.e., the one the most likely to be colliding. A similar technique is used in [NK00], where the segments are sorted according to some probability of being collision-free.

Algorithm TEST-PATH(τ)
1. While U is not empty do
   1.1. $u \leftarrow$ extract(U)
   1.2. If TEST-SEGMENT(u) = *collision* then
      1.2.1. Remove u from the roadmap
      1.2.2. Return *nil*
   1.3. If u is not marked *safe*, then re-insert it in U
2. Return τ

Each loop of Step 1 tests the segment u that is in first position in U at one additional level of resolution. This segment is removed from U. It is re-inserted in U if TEST-SEGMENT(u) neither detects a collision, nor marks u as *safe*. If u is re-inserted in U, it may not be in first position, since the quantity $\varepsilon(u)$ has been divided by 2. TEST-PATH terminates when a collision is detected – then the colliding segment is removed – or when all segments have been marked *safe* (i.e., U is empty) – then the path τ is returned.

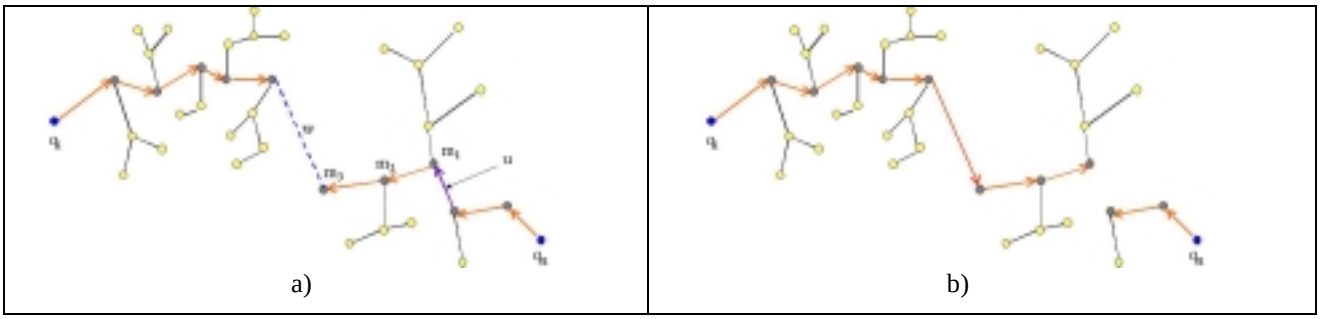


Figure 5: Transfer of milestones between trees

The removal of a segment u disconnects again the roadmap into two trees. If u is the bridge that was created by CONNECT, the two trees return to their previous state (except for some *Unsafe* sets, whose contents may have changed). Otherwise, the removal of u results in transferring milestones from one tree to the other. Assume that u is in T_{goal}, as illustrated in Figure 5a, where $w \neq u$ denotes the bridge added by CONNECT. The milestones m_1, …, m_r between u and w ($r = 3$ in Figure 5) and their children in T_{goal} are transferred to T_{init} as shown in Figure 5b. The connections between transferred milestones remain the same, except those between m_1, …, m_r, which are inverted. So, no milestone is ever removed from the roadmap. The collision-checking work done along all segments, except u, is saved in the *Unsafe* sets. Hence, if one of them later lies on another candidate path, the work previously done is not repeated.

2.3.5. Implementation details

Collision checking. SBL's collision checker is PQP [GLM96]. Each obstacle and robot link is described by a collection of triangles representing its surface. PQP pre-computes a bounding hierarchical representation of each object using oriented-bounding boxes. No other pre-computation is done.

PQP can be used to compute the distance between two objects, or to only check whether a collision occurs. In the current SBL's implementation, it is used as a pure collision checker. To check whether a segment u connecting two milestones is collision-free, we test its midpoint (this point has a high probability of colliding if the segment collides) and, if this point is collision-free, we recursively break the segment into halves and test the midpoint of each half. The recursion ends either when a collision is detected, or when the distance between two consecutive tested points is less than some predefined ε, in which case u is marked *safe*.

SBL associates an index $\kappa(u)$ with each segment u (including the bridge). $\kappa(u)$ is an integer indicating the resolution at which u has already been tested. If $\kappa(u) = 0$, then only the two endpoints of u (both are milestones) have been tested. If $\kappa(u) = 1$, then the two endpoints and the midpoint of u have been tested. More generally, for any $\kappa(u)$, $2^{\kappa(u)}+1$ equally distant points of u have been tested. Let $\lambda(u)$ be the length of u. If $2^{-\kappa(u)}\,\lambda(u) < \varepsilon$, then u is marked *safe*. The index of each new segment is initialized to 0.

Let $\sigma(u,j)$ designate the set of points in u that must have already tested collision-free in order for $\kappa(u)$ to take the value j. The implemented algorithm TEST-SEGMENT(u) increases $\kappa(u)$ by 1:

Algorithm TEST-SEGMENT(u)
1. $j \leftarrow \kappa(u)$
2. For every point $q \in \sigma(u,j+1)\backslash\sigma(u,j)$, if $D(q) = 0$ then return *collision*
3. If $2^{-(j+1)}\lambda(u) < \varepsilon$, then mark u as *safe*, else $\kappa(u) \leftarrow j+1$

For each segment u that is not marked *safe*, the value of $2^{-\kappa(u)}\,\lambda(u)$ is cached in the data structure representing u. The priority queue U maintained by the implemented TEST-PATH consists of all the segments in $\{u_1, u_2, \ldots, u_p\}$ that are not marked *safe*, sorted in decreasing order of $2^{-\kappa(u_i)}\,\lambda(u_i)$. Each loop of TEST-PATH increases the index of the first segment in U by 1.

Spatial Indexing. SBL spatially indexes every milestone of T_{init} (resp. T_{goal}) in an h-dimensional array A_{init} (resp. A_{goal}). Both arrays partition the subspace defined by h dimensions of C (in our implementation, $h = 2$) into the same grid of equally sized cells. Whenever a new milestone q is installed in a tree, the appropriate cell of the corresponding array is updated to contain q. When a milestone is transferred from one tree into the other, the two arrays are updated accordingly. A_{init} and A_{goal} are used at Step 2.1 of EXPAND where we pick a milestone m from one tree T with probability $1/\eta(m)$. Rather than maintaining the density $\eta(m)$ around each milestone, we do the following. Assume that $T = T_{init}$. We first pick a non-empty cell of A_{init}, then a milestone from this cell.

Hence, the probability to pick a certain milestone is greater if this milestone lies in a cell of A_{init} containing fewer milestones. This technique is fast and results in a good diffusion of milestones in F along the h selected dimensions. To ensure diffusion along all dimensions of C, we pick the h dimensions at random and we periodically change them. Each change requires re-constructing the arrays A_{init} and A_{goal}, but the total cost of this operation is negligible relative to collision tests.

CONNECT also uses A_{init} and A_{goal} at Step 2 to identify the milestone m' that will be connected to the newly added milestone m. Our implementation of CONNECT tries two connections: (1) instead of selecting m' as the closest milestone to m in the other tree, it picks m' to be the closest milestone in the same cell as m, but in the other array (m and m' are then only guaranteed to be close to each other along h dimensions); (2) it picks m' uniformly at random in the other tree. Our experiments show that this technique is faster on average than connecting m to the closest milestone. (The "closest-milestone" heuristics often delays the finding of some easy connections.)

Path optimization. We added a simple path optimizer to SBL to remove blatant jerks from paths. This optimizer takes a path τ as input and performs the following operation a number (typically, 10 to 20) of times: pick two points q and q' in τ at random and, if the straight-line segment joining them tests collision-free, replace the portion of τ between q and q' by this segment.

2.4. Experimental results

SBL is written in C++ and compiled with g++ 2.5.3. The running times given below were gathered on a 1-GHz Pentium III processor with 1 GB of main memory running Linux. The distance threshold ρ was set to 0.15 and the collision-checking resolution ε to 0.01. Each of the arrays A_{init} and A_{goal} had size 10×10. The two dimensions of these arrays were changed every 50 milestones. The pre-computation time of PQP is not included in the planner's running time, but is indicated separately below. The value of ε was determined to ensure that the probability of SBL generating an incorrect path is negligible in the geometric environments in which we did our experiments. It is similar to the one used in some previous papers (e.g., [BK00]). However, it would be preferable to perform the rigorous collision test defined in Subsection 2.1.1.

2.4.1. Environments

The environments in Figure 2 are among those we used to test SBL. In each example, we let n_{rob} and n_{obs} denote the numbers of triangles in the geometric models of the robot and the obstacles, respectively.

- The robot in Figure 2a is a 6-dof robot arm equipped with a welding gun. The obstacle is a car body that contains both thin (surface-like) and narrow (line-like) components. The robot has much maneuvering space, except when its tool gets close to the car body. Here, n_{rob} = 5,000 and n_{obs} = 21,000.

- The robot in Figure 2b is another 6-dof arm. The environment was designed to produce narrow passages in configuration space. Such passages (e.g., the hole in the wall) require several dofs to move, or stay almost still, in a tightly coordinated way. Here, n_{rob} = 3,000 and n_{obs} = 100.

- In Figure 2c, the robot must transfer a large metal sheet from a table to a press. n_{rob} = 5,000; n_{obs} = 83,000.

- In Figure 2d, the robot loads/unloads parts into/from a lathe. The regions close to the initial and final configurations are highly constrained. n_{rob} = 3,000 and n_{obs} = 50,000.

- Figure 2e shows another environment designed to create narrow passages. n_{rob} = 3,000; n_{obs} = 50.

SBL computes robot paths in configuration space, but in Figure 2, we show the curves traced by the center-point of the robot's end-effector in the workspace. In each environment, the dark-color (red) and light-color (yellow) curves respectively correspond to paths before and after optimization.

The time used by PQP to pre-compute the bounding hierarchies goes from 0.19 seconds for the environment with the fewest triangles (e) to 3.87 seconds for the environment with the largest number of triangles (c).

Running Time(secs)	*Milestones in Roadmap*	*Milestones in Path*	*Total Nr of Collision Checks*	*Collision Checks on the Path*	*Sampled Milestones*	*Comput. Time for Coll-Check (secs)*
0.36	112	9	934	247	174	0.36
1.19	216	21	3170	602	334	1.17
0.4	95	9	884	234	148	0.4
0.64	167	18	1701	461	265	0.64
1.09	200	10	2625	272	311	1.06
0.78	178	20	2038	520	260	0.76
0.51	150	14	1307	411	239	0.5
0.46	67	15	1112	377	100	0.45
0.46	104	16	1213	420	156	0.46
0.63	194	13	1499	322	329	0.62

Figure 6: Experimental results in the environment of Figure 2a

2.4.2. Basic performance evaluation

Figure 6 shows results for the environment of Figure 2a for 10 runs with the same query configurations, but with different seeds of the random number generator. The running times are without optimization. The non-optimized path shown in Figure 2a corresponds to the third line in the table of Figure 6. In that example, the optimized path took an additional 0.19 seconds to generate.

Example	*Running Time (s)*	*Milestones in Roadmap*	*Milestones in Path*	*Total Number of Collision Checks*	*Collision Checks on the Path*	*Sampled Milestones*	*Computational Time for Coll-Check (s)*	*Std. Deviation for running time*	*PQP Pre-comp. time (s)*
a	0.60	159	13	1483	342	245	0.58	0.38	1.65
b	4.45	1609	39	11211	411	7832	4.21	2.48	0.31
c	4.42	1405	24	7267	277	3769	4.17	1.86	3.87
d	0.17	33	10	406	124	47	0.17	0.07	2.65
e	6.99	4160	44	12228	447	6990	6.30	3.55	0.19

Figure 7: Statistics on experiments in the environments of Figure 2

The table in Figure 7 shows averages for 100 runs of SBL on five examples, each in a different environment of Figure 2. The last column indicates the standard deviation on the running times. In all cases, SBL successfully found a path; there was no failure (the maximal number of milestones s was set to 10,000). In all runs, a significant fraction of the collision checks were made on the solution path. As noticed in [BK00], these tests cannot be avoided.

On several examples, we collected statistics for different values of the parameter ρ ranging between 0.1 and 0.3. They did not reveal major variations in the planner's running time. We also tried indexing arrays of resolutions other than 10×10, including three-dimensional arrays; performance results were not significantly different.

2.4.3. Experimental convergence rate

Figure 8 shows two diagrams respectively established for the examples of Figures 2c and 2d. Each diagram was generated by running SBL with increasing values of the maximum number of milestones, s, from very small ones to larger ones. The values are indicated on the horizontal axis of the diagram. For each value of s, we ran SBL 200 times with different seeds, and we counted the number of failures (vertical axis of the diagram). When s is very small, SBL fails consistently; when it is sufficiently large, its success rate is 100%. The transition between consistent failures and consistent successes is rather fast, which is coherent with the theoretical result that a PRM planner has a fast convergence rate in the number of milestones [Hsu00].

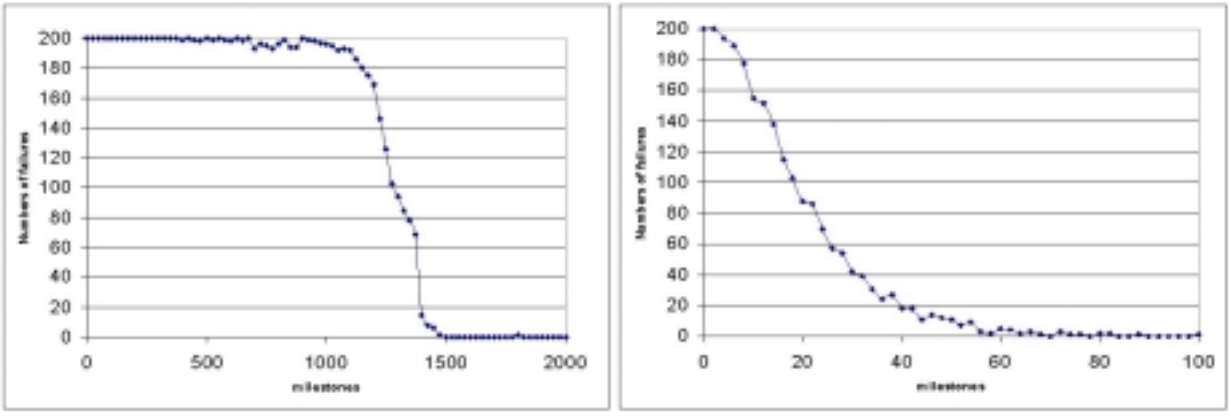


Figure 8: Experimental convergence rate of SBL on problems c and d, respectively

2.4.4. Comparative performance evaluation

To assess the efficiency of the lazy collision-checking strategy, we have implemented a version of our planner that fully tests every connection before inserting it in the roadmap. This planner is similar to the one presented in [Hsu00]. Note, however, that our two planners do not exactly generate the same milestones, even when they use the same seed for the random number generator. Indeed, while SBL considers any collision-free configuration *q* picked in the neighborhood of a milestone *m* as a new milestone (Step 2.2 of EXPAND), the full collision-check planner also requires that the connection between *m* and *q* be collision-free. Moreover, in the second planner no milestone is ever transferred from one tree to the other.

Example	*Running Time (s)*	*Milestones in Roadmap*	*Milestones in Path*	*Total Nr of Collision Checks*	*Collision Checks on the Path*	*Sampled Milestones*	*Comput. Time for Coll-Check (s)*	*Std. Deviation for running time*
a	2.82	22	5	7425	173	83	2.81	3.01
b	106.20	3388	32	300060	421	9504	105.56	59.30
c	18.46	771	16	38975	219	3793	18.35	15.34
d	1.03	29	9	2440	123	46	1.02	0.70
e	293.77	6737	24	666084	300	11971	292.40	122.75

Figure 9: Experimental results for the full collision-check planner

Figure 9 shows results with this full collision-check planner, on the same five problems as above. Again, the maximal number of milestones *s* was set to 10,000 and the results are averages over 100 runs. The average running times (and numbers of collision checks) for the lazy planner are significantly smaller than for the full collision-check planner. The speed-up is greater on problems b, c, and e, which have the longest running times. Moreover, in example b, there were some cases in which the full collision-check planner was not able to find a path in 10,000 iterations. Thus, in this example, the lazy planner compares more favorably than indicated in the tables (the averages in Figure 9 were computed over the successful runs only). These results cannot be compared to those in [BK00], where the improvement was measured relative to a *multi-query* planner, which must pre-compute a large roadmap to cover the entire free space.

3. Multi-Robot Planning

We now turn to describing the application of SBL to compute coordinated paths for multiple robots sharing a subset of their respective workspaces. Multi-robot welding stations in automotive body shops (Figure 1) are a good example of such multi-robot systems, with a number of dofs ranging between 20 and 60.

3.1. Centralized vs. decoupled planning

There exist two classic approaches to motion planning for multiple interacting robots [Lat91], centralized and decoupled planning:

(1) *Centralized planning* consists of considering the robots as if they were forming a single multi-arm robot, by encoding all their dofs in a single joint configuration space C and searching C for a collision-free path. The number of dimensions of C is equal to the total number of dofs of the robots. A collision-free path in C, if one exists, not only describes the individual path to be followed by each robot, but also how the robots are to be coordinated. Centralized planning has not been considered a practical approach because it leads to searching configuration spaces of high dimensionality. Most proposed techniques are based on ad-hoc and incomplete heuristics, for example potential field techniques [BLL92, Buc89, Tou86, War90], which may not be reliable enough to be widely useful. Complete centralized planning algorithms have only been proposed for very simple robotic systems, e.g., the coordination of two discs among polygonal obstacles [SS83]. PRM-based centralized planning has previously been used for disassembly planning [SRA01].

(2) *Decoupled planning* is a two-phase approach. First, a collision-free path is generated for each robot by considering only the obstacles in the environment and ignoring the other robots. The second phase, called *velocity tuning*, selects the relative velocities of the robots along their respective paths to avoid collision among them [AdB+99, KZ86, OLP89]. Compared to centralized planning, decoupled planning leads to searching smaller-dimensional spaces. But it is inherently incomplete, even if the algorithms used in the first and second phases are complete. Indeed, the second phase may fail because the paths generated in the first phase cannot be coordinated without collision between robots, while this coordination would have been possible if other paths had been selected. An alternative to velocity tuning, called *prioritized planning*, is proposed in [ELP87]. It consists of processing the robots in some predefined order and planning the path of each robot by treating the robots whose paths have already been planned as moving obstacles. A resolution-complete algorithm for coordinating car-like robots along independently planned paths is given in [LLS99]. Other coordination schemes, e.g., plan-merging techniques [ARIS95], have also been proposed.

Multi-robot planning has attracted considerable research, and the above bibliographical references are not exhaustive. However, most proposed planners fall into either one of the above two approaches.

3.2. Velocity tuning

Velocity tuning is one possible technique to coordinate independently generated paths, by searching a "reduced" configuration space. To illustrate, consider two robots whose respective paths, τ_1 and τ_2, have been independently planned. By forcing the robots to move along these paths, we reduce the number of dofs of each robot to 1, hence the dimension of their joint configuration space to 2. Let each path τ_i (i = 1 or 2) be parameterized by its curvilinear length $s_i \in [0,L_i]$ from the initial configuration; so, L_i denotes the total path length. The set $P = [0,L_1] \times [0,L_2]$ represents the reduced configuration space of the two robots, called the *coordination space*. Each point $(s_1,s_2) \in P$ defines a placement of the two robots at their respective configurations $\tau_1(s_1)$ and $\tau_2(s_2)$. This point is collision-free if at this placement the two robots do not collide with each other. (Collisions with obstacles in the environment were already taken care of during the generation of τ_1 and τ_2.) A path joining the point (0,0) – where both robots are at their initial configurations – to the point (L_1,L_2) – where they are at their goal configurations – in the collision-free subset of P defines a valid coordination of the two robots along τ_1 and τ_2 [OLP89]; it sets the relative velocities of the robots along their respective paths. Note that the coordinated path may not be monotonic along any of the dimensions of P. If it is non-monotonic along s_i (i = 1 or 2), then for a while the i^{th} robot will move backward along τ_i. Such motion may be required to provide maneuvering space to the second robot. An unfortunate choice of τ_1 and τ_2, however, may lead the points (0,0) and (L_1,L_2) to lie in two distinct connected component of the free subset of P.

If there are $p > 2$ robots, we may generate a coordinated collision-free path in the p-dimensional coordination space P, where the i^{th} axis encodes the curvilinear length along the path of the i^{th} robot, from the point $(0,\ldots,0)$ to the point $(L_1,\ldots,L_p)$. We term this approach to velocity tuning *global coordination*. An alternative, *pairwise coordination*, consists of planning p-1 paths in a series of p-1 two-dimensional coordination spaces $P_2,\ldots,P_p$. The axes of P_2 encode the lengths along the paths of the 1st and 2nd robots, so that a collision-free path in P_2 defines a valid coordination of these two robots. One axis of P_3 encodes the length along the path of the 3rd robot, while the other axis encodes the length along the coordinated path of the 1st and 2nd robots. Hence, each point in P_3 determines a placement of the first three robots, and a collision-free path in P_3 defines a valid coordination of these robots, etc. A decoupled planning algorithm based on global coordination is inherently incomplete, but less so than one based on pairwise coordination, since a specific path selected in the coordination space P_i may result into a space P_{i+1} with no collision-free path between $(0,\ldots,0)$ and $(L_1,\ldots L_{i+1})$. In the past, pairwise coordination has been more widely used than global coordination, since it only requires planning in two-dimensional coordination spaces.

3.3. SBL-based multi-robot planners

We have developed three multi-robot planners respectively based on centralized planning, decoupled planning with global coordination, and decoupled planning with pairwise coordination:

(1) The first planner (centralized planning) is essentially SBL. In that case, SBL directly plans a path in the joint configuration space of the robots.

(2) The second planner (decoupled planning with global coordination) makes p+1 calls to SBL, where p is the number of robots: p calls to plan the path of each robot (ignoring the other robots) in its configuration space; then, another call to SBL to plan a collision-free path in the coordination space $P = [0,L_1] \times \ldots \times [0,L_p]$. The roadmap in P is built as described in Subsection 2.3 by growing two trees of milestones respectively rooted at $(0,\ldots,0)$ and $(L_1,\ldots,L_p)$. A path in this roadmap may not be monotonic along any of the axes of P. The collision test of a point in P is done by calling the collision checker on every pair of links belonging to two distinct robots.

(3) The third planner (decoupled planning with pairwise coordination) makes 2p-1 calls to SBL: p calls to plan the path of each robot in its own configuration space; then, p-1 calls to plan collision-free paths in the two-dimensional coordination spaces P_2 through P_p. The collision test of a point in P_i is done by calling the checker on every pair of links consisting of one link of the i^{th} robot and a link of one of the robots 1 to i-1.

All three planners use the path optimizer described in Subsection 2.3.5. In the second and third planners, this optimizer is called after the robot paths have been coordinated. Indeed, optimizing the individual paths before they are coordinated would reduce the flexibility left for coordinating them, hence increase the incompleteness of the planners.

Before showing and analyzing experimental results obtained with these three planners, it is important to discuss what a failure means for each of them. Since centralized planning is a complete approach to multi-robot planning and SBL is only probabilistically complete, a failure of the SBL-based centralized planner can have only two causes: (1) there is no solution to the problem, or (2) the planner failed to find a solution within the maximum number s of milestones specified as input. On the other hand, decoupled planning is inherently incomplete. Hence, a failure of any of the two SBL-based decoupled planners can have four causes: (1) there is no solution to the problem, (2) SBL failed to find a path for one of the robots, (3) it found a path for each robot, but there is no valid coordination of these paths (i.e., there exists no collision-free path in the coordination space corresponding to these paths), or (4) there is a valid coordination of these paths, but SBL failed to find it. The diagrams of Figure 8 show that if SBL returns a solution to a given problem for a certain value of s, then we can easily make SBL succeed consistently on this problem by increasing s. This is what we did in the experiments reported below. For each run of SBL, we set s large enough to guarantee (with very high probability) that no failure is caused by the fact that SBL is only probabilistic complete.

3.4. Experimental results

We have experimented with the three planners on several examples in the environment of Figure 10, which represents a welding station similar to those in Figure 1. This station contains six robots with six dofs each. In order to keep our planners as general as possible, we did not take advantage of certain properties of this environment. For example, the planners assume that collisions can occur between any two links of any two robots, while it is clear that many pairs of links cannot collide. So, since each robot consists of 6 rigid links, testing that no two robots collide at a given configuration requires running the collision checker on 540 pairs of links. Fortunately, when two links are sufficiently far apart, the pre-computed bounding hierarchies lead the checker to quickly detect that they cannot possibly collide.

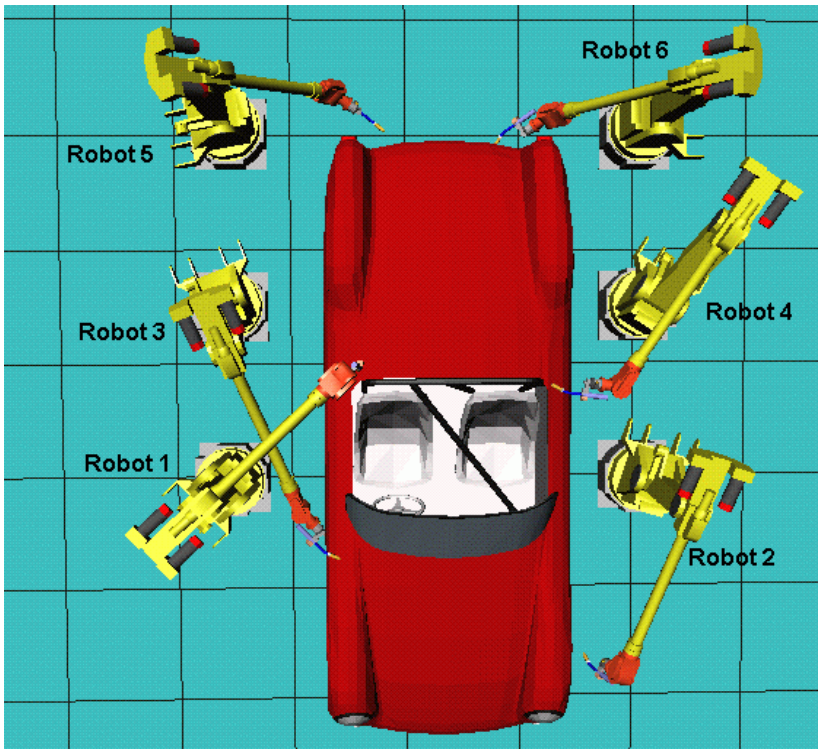


Figure 10: Six-robot environment

3.4.1. Results with centralized planner

Figures 11-13 show the initial and goal configurations of the robots for three problems named I, II, and III. The table of Figure 14 shows averages collected over 100 runs of the centralized planner on problems I, II, and III, with 2, 4, and 6 robots. The robots are identified 1, 2, …, 6, as shown in Figure 10. A problem for 2 (or 4) robots is defined as shown in Figures 11-13, but restricted to the robots 1 and 2 (or 1, 2, 3, and 4). In all $100 \times 3 \times 3 = 900$ runs, the planner successfully returned a path in a satisfactory amount of time. Figure 15 lists the largest and smallest running times and numbers of milestones over the 100 runs for each of the 3×3 problems. Figure 16 shows a series of snapshots along a path computed for problem III with two robots.

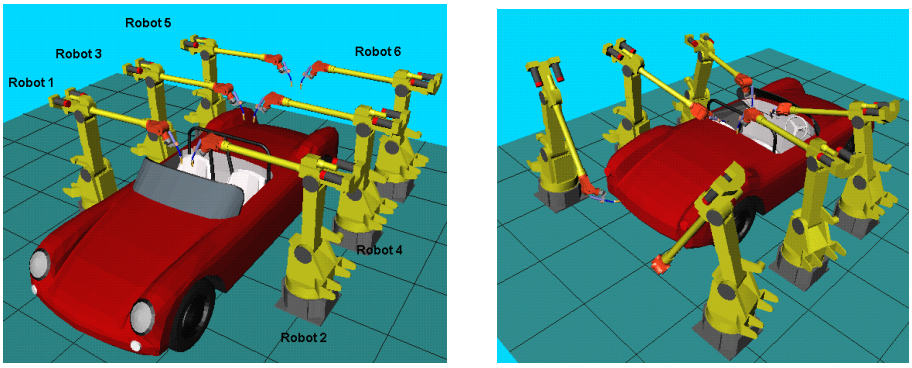


Figure 11: Problem I

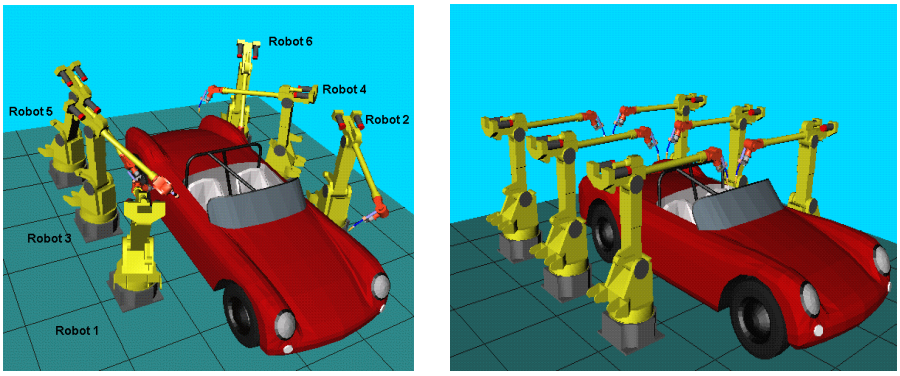


Figure 12: Problem II

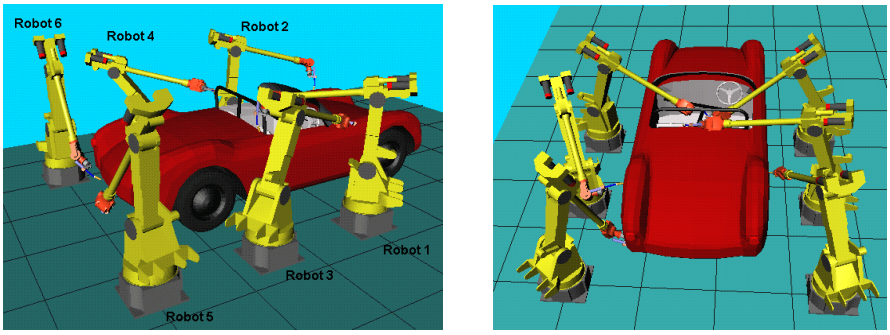


Figure 13: Problem III

Problem	*Running time (secs)*	*Milestones in Roadmap*	*Milestones in Path*	*Total Nr of Collision Checks*	*Collision Checks on the Path*	*Sampled Milestones*	*Comput. Time for Coll-Check (secs)*	*Std. Deviation for running time*
PI- 2 Robots	0.26	11	4	242	58	18	0.26	0.52
PII- 2 Robots	0.25	11	5	248	76	13	0.25	0.17
PIII-2 Robots	2.44	191	17	2356	243	718	2.41	1.57
PI-4 Robots	3.97	62	7	1015	106	193	3.96	5.67
PII-4 Robots	3.94	56	10	968	166	112	3.93	2.40
PIII-4 Robots	30.82	841	32	8895	542	2945	30.57	15.55
PI-6 Robots	28.91	322	14	3599	212	1083	28.82	28.91
PII-6 Robots	59.65	882	30	6891	533	1981	59.41	31.08
PIII-6 Robots	442.85	5648	91	47384	1525	24511	439.39	170.46

Figure 14: Average measurements for centralized planner

Problem	Time		Milestones in Roadmap	
	Min	Max	Min	Max
PI-2 Robots	0.05	4.93	3	171
PI-4 Robots	0.23	28.75	3	479
PI-6 Robots	0.85	122.55	3	1703
PII-2 Robots	0.09	1.02	6	81
PII-4 Robots	0.85	10.63	12	197
PII-6 Robots	11.28	127.19	73	1878
PIII-2 Robots	0.42	7.60	22	579
PIII-4 Robots	8.31	92.09	188	2577
PIII-6 Robots	322.52	564.35	4378	6917

Figure 15: Largest/smallest running times and numbers of milestones for centralized planner

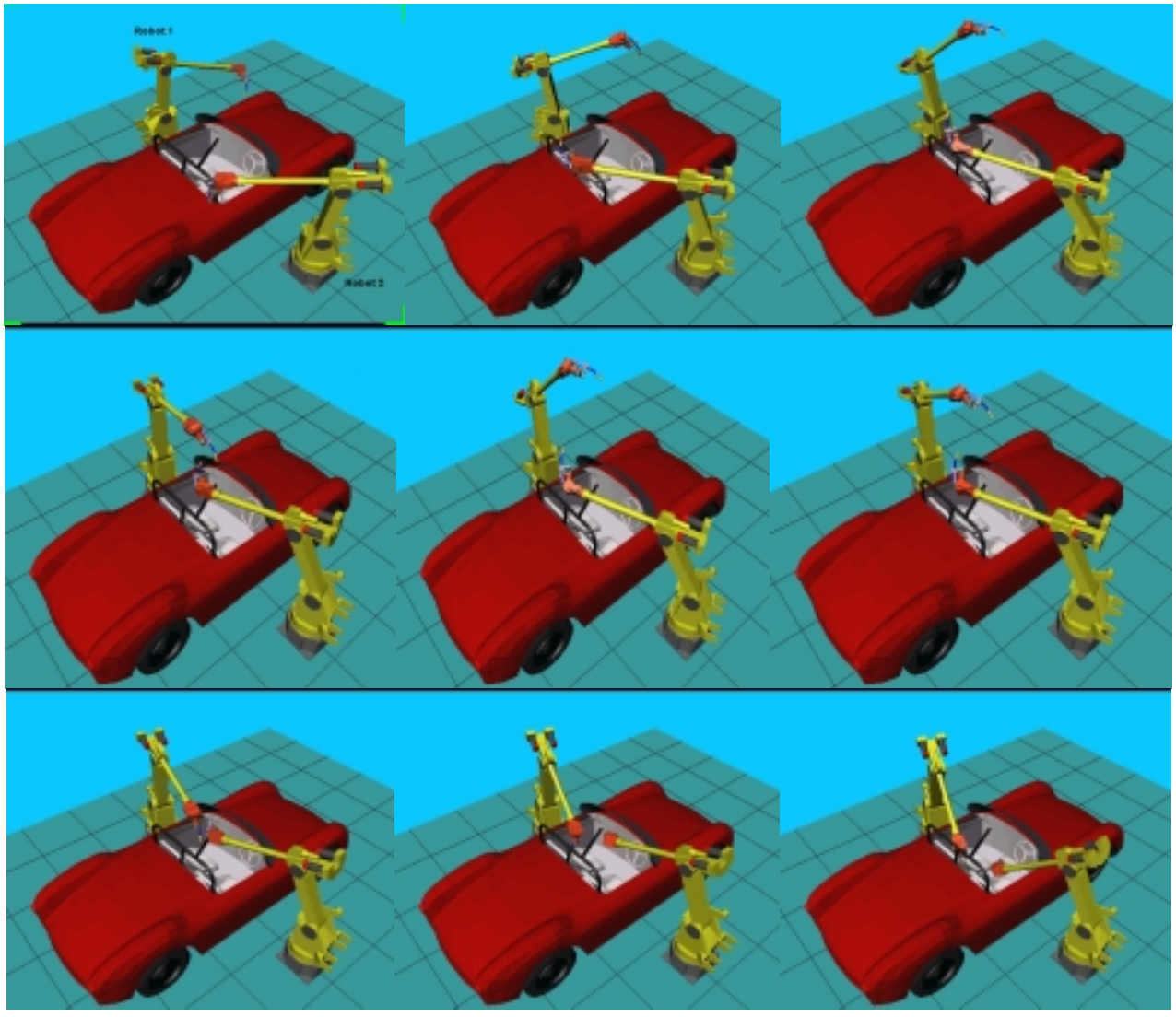


Figure 16: Snapshots along a path computed by the centralized planner for problem III with two robots

The increase in the running times observed when the number of robots goes from 2 to 4 to 6 is caused in part by the quadratic increase in the number of pairs of links that may have to be tested at each collision-checking operation, and in part by the greater difficulty of the problems due to the constraints imposed by the additional robots upon the motions of the other robots. Note that the time per collision check increases from 0.001 seconds for 2 robots, to less than 0.004 seconds for 4 robots, to about 0.0085 seconds for 6 robots. So, it increases more slowly than the number of pairs of links tested. The reason is that each collision check involves testing each link against the environment and each pair of links from two different robots against each other. On average, the second type of test is less costly than the first, because the model of the environment contains more triangles than that of a link, and also because many two links are far apart (especially when the number of robots increases). While the number of tests of the first type is linear in the number of robots, the number of tests of the second type is quadratic.

We also created a version of the centralized planner specific to the environment of Figure 10. In this version, the planner only tests pairs of links that can possibly collide, and it orders the tests from the end-effector of a robot toward its base, since the links the furthest away from the base are the most likely to collide. For the most complex problem (problem III with 6 robots), the average running time of the planner was reduced from 443 to 323 seconds.

3.4.2. Results with decoupled planners

Unlike the centralized planner, the decoupled planners failed a number of times on the above 3×3 problems. For each run of SBL, the maximal number of milestones was set large (50,000). In particular, this means that each of the decoupled planners had up to 50,000 milestones to generate each robot path. The decoupled planner with global coordination had up to 50,000 milestones to search the coordination space. The decoupled planner with pairwise coordination had up to 50,000 milestones to search *each* of the two-dimensional coordination space.

Planner	PROBLEM I						PROBLEM II						PROBLEM III					
	2 Robots		4 Robots		6 Robots		2 Robots		4 Robots		6 Robots		2 Robots		4 Robots		6 Robots	
	Time(s)	Failures	Time(s)	Failures	Time(s)	Failures	Time(s)	Failures	Time(s)	Failures	Time(s)	Failures	Time(s)	Failures	Time(s)	Failures	Time(s)	Failures
Centralized	0.26	0	3.97	0	28.91	0	0.25	0	3.94	0	59.65	0	2.44	0	30.81	0	442.85	0
Dec. Global	0.22	1	2.74	3	29.53	7	0.37	2	6.59	4	65.45	6	4.32	5	16.23	6	267.81	13
Dec. Pairwise	0.30	3	4.85	5	19.23	9	0.42	3	5.63	7	28.92	6	3.42	9	25.35	13	182.63	17

Figure 17: Comparison of centralized and decoupled planners

The table of Figure 17 lists the number of failures and the average running times of the two decoupled planners over 20 runs on each of problems I, II, and III, with 2, 4, and 6 robots. (The average running times are computed only over the successful runs.) For comparison, we also include the averages for the centralized planner (established over 100 runs). The number of failures of each decoupled planner was relatively small for the three problems with 2 robots. But their rate of failures for the problems with 6 robots ranged from 30% to 75%, with pairwise coordination being more unreliable than global coordination. The running times are not easily comparable. For the more complex Problem III with 4 and 6 robots, decoupled planning (when it succeeds) is only marginally faster than centralized planning.

The decoupled planners never failed while generating individual robot paths. All their failures happened when velocity tuning in a coordination space failed after generating 50,000 milestones. In every successful run of a decoupled planner, the number of milestones in each of the roadmaps built by this planner was much smaller than 50,000 (at least 10 times smaller), indicating that with very high probability the planner's failures in other runs were caused by the incompleteness of the decoupled planning approach, not by SBL's probabilistic completeness.

These results invalidate the prevailing assumption that the loss of completeness in performing decoupled planning is not very significant in practice and worth the computational gain. Our results indicate that

centralized planning may be a much more desirable approach, and that the existence of efficient PRM planners such as SBL makes this approach technically feasible. Clearly, however, there are problems where centralized planning is not suitable, or even possible – e.g., problems where mobile robots with onboard computing and limited communication channels operate in the same environments. For such problems, decoupled planning remains the preferred approach.

4. Conclusion

The planner's design and the experimental results presented in this paper demonstrate that a PRM planner combining a connection strategy based on delayed collision checking with a single-query bi-directional sampling strategy can solve path-planning problems of practical interest (i.e., with realistic complexity) in times ranging from fractions of a second to a few seconds for 1 to 4 robots, and from seconds to a few minutes for 4 to 6 robots. Our results on problems requiring tight coordination among several robots clearly show that the centralized approach to multi-robot planning is preferable over the decoupled approach.

Several relatively straightforward improvements are still possible. For instance, we could use a weighted L_∞ metric in configuration space that better reflects the magnitude of the robot's displacement caused by a small variation of each dof. We could also use PQP algorithm for computing distances rather than as a pure collision checker, in order to rigorously test connections between milestones; this could also significantly reduce the number of calls to PQP. In multi-robot systems, we could speed-up collision checking by using general space indexing techniques and environment-specific simplifications to reduce the number of pairs of links tested.

Our main current goal is to extend SBL to facilitate the programming of multi-robot welding stations in automotive body shops (Figure 1). One important extension will be to plan for multiple goals. Each robot must perform welding operations at various locations on a car body, but the ordering of these locations is not fully specified. This will require computing an optimized tour of the welding locations to be visited by each robot. This is a variant of the classical Traveling Salesman Problem, where the distance between two locations is not given and, instead, must be computed by the underlying PRM planner. Clearly, if there are r locations to visit, we do not want to invoke our planner $O(r^2)$ times; so, a better method must be found. Prior work on multi-goal planning problems can be found in [BSB99, SR00, WHW99].

Acknowledgements: This research was conducted in the Computer Science Dept. at Stanford University. It was funded by grants from General Motors Research and ABB. G. Sánchez's stay at Stanford was partially supported by ITESM (Campus Cuernavaca) and a fellowship from CONACyT. This paper has greatly benefited from discussions with H. González-Baños, C. Guestrin, D. Hsu, L. Kavraki, and F. Prinz.. The authors also thank S. Holland, W. Ryback, and C. Wampler from General Motors for introducing the multi-robot welding problem to them and discussing several aspects of this research. PQP was made available by S. Gottschalk, M. Lin, and D. Manocha from the Computer Science Dept. at the University of North-Carolina in Chapel Hill.

References

[Aga97] P.K. Agarwal. Range Searching. In *Handbook of Discrete and Computational Geometry*, J.E. Goodman and J. O'Rourke (eds.), CRC Press, pp. 575-598, 1997.

[Ahu94] J.M. Ahuactzin. Le Fil d'Ariane. *Une méthode de planification générale. Application à la planification de trajectoires*. PhD Thesis, National Polytechnic Institute of Grenoble, France, 1994.

[AG99] J.M. Ahuactzin and K.K. Gupta. The Kinematic Roadmap: A Motion Planning Based Global Approach for Inverse Kinematics of Redundant Robots. *IEEE Tr. On Robotics and Automation*, 15(4):653-669, 1999.

[ARIS95] R. Alami, F. Robert, F.F. Ingrand, and S. Suzuki. Multi-Robot Cooperation Through Incremental Plan-Merging. *Proc. IEEE Int. Conf. on Robotics and Automation*, Nagoya, Japan, pp. 2573-2578, 1995

[ABD+98] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. OBPRM: An Obstacle-Based PRM for 3D Workspace. In P.K. Agarwal et al. (eds.), *Robotics: The Algorithmic Perspective*, A K Peters, Natick, MA, pp. 155-168, 1998.

[ABD+00] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. Choosing Good Distance Metrics and Local Planners for Probabilistic Roadmap Methods. *IEEE Tr. on Robotics and Automation*, 16(4):442-447, August 2000.

[AW96] N.M. Amato and Y. Wu. A Randomized Roadmap Method for Path and Manipulation Planning. *Proc. IEEE Int. Conf. on Robotics and Automation*, Minneapolis, MN, pp. 113-120, 1996.

[ABG+02] M.S. Apaydin, D. Brutlag, C. Guestrin, D. Hsu, and J.C. Latombe. Stochastic Roadmap Simulation: An Efficient Representation and Algorithm for Analyzing Molecular Motion. *Proc. 6th Annual Conf. on Research in Computational Molecular Biology (RECOMB'02)*, Washington D.C., April 2002.

[AdB+99] B. Aronov, M. De Berg, A. F. Van der Stappen, P. Švestka, and J. Vleugels. Motion planning for multiple robots. *Discrete and Computational Geometry*, 22:505-525, 1999.

[Bag96] B. Baginski. The Z^3-Method for Fast Path Planning in Dynamic Environments. *Proc. IASTED Conf. Applications of Control and Robotics*, Orlando, FL, pp. 47-52, 1996.

[Bag97] B. Baginski. Local Motion Planning for Manipulators Based on Shrinking and Growing Geometry Models. *Proc. IEEE Int. Conf. on Robotics and Automation*, Minneapolis, MN, pp. 3303-3308, 1997.

[Bar90] D. Baraff. Curved Surfaces and Coherence of Non-Penetrating Rigid Body Simulation. *ACM Computer Graphics*, 24(4):19-28, 1990.

[BKL+97] J. Barraquand, L.E. Kavraki, J.C. Latombe, T.Y. Li, R. Motwani, and P. Raghavan. A Random Sampling Scheme for Path Planning. *Int. J. of Robotics Research*, 16(6):759-774, 1997.

[BL91] J. Barraquand and J.C. Latombe. Robot Motion Planning: A Distributed Representation Approach. *Int. J. of Robotics Research*, 10(6):628-649, 1991.

[BLL92] J. Barraquand, B. Langlois, and J.C. Latombe. Numerical Potential Field Techniques for Robot Path Planning. *IEEE Tr. on Systems, Man, and Cybernetics*, 22(2): 224-241, March 1992.

[BK00] R. Bohlin and L.E. Kavraki Path Planning Using Lazy PRM. *Proc. IEEE Int. Conf. Robotics and Automa*tion, San Francisco, CA, April 2000.

[BSB99] M. Bonert, L.H. Shu, and B. Benhabib. Motion Planning for Multi-Robot Assembly Systems. *Proc. ASME Design Engineering Technical Conf.*, Las Vegas, NE, Sept. 1999.

[BOvdS99] V. Boor, M.H. Overmars, A.F. van der Strappen. The Gaussian Sampling Strategy for Probabilistic Roadmap Planners. *Proc. IEEE Int. Conf. on Robotics and Automation*, Detroit, MI, pp. 1018-1023, 1999.

[Buc89] S.J. Buckley. Fast Motion Planning for Multiple Moving Robots. *Proc. IEEE Int. Conf. on Robotics and Automation*, Scottsdale, AZ, pp. 1419-1424, 1989.

[Cas01] A. Casal. *Reconfiguration Planning for Modular Self-Reconfigurable Robots*. Aeronautics & Astronautics Dept., Stanford University, Stanford, CA, December 2001.

[CL95] H. Chang and T.Y. Li. Assemblability Maintainability Study with Motion Planning. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1012-1019, 1995.

[CLS00] M.G. Choi, J. Lee and S.Y. Shin, *A Probabilistic Approach to Planning Biped Locomotion with Prescribed Motions*. Tech. Rep. CS-TR-2000-162, Computer Science Dept., Korea Advanced Institute of Science & Technology, 2000.

[CLMP95] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-Collide: An Interactive and Exact Collision Detection System for Large Scale Environments. *Proc. ACM Interactive 3D Graphics Conf.*, pp. 189-196, 1995.

[ELP87] M.A. Erdmann and T. Lozano-Perez. On Multiple Moving Objects. *Algorithmica*. 2(4):477-521, 1987.

[Fav89] B. Faverjon. Hierarchical Object Models for Efficient Anti-Collision Algorithms. *Proc. IEEE Int. Conf. on Robotics and Automation*, Scottsdale, AZ, pp. 333-340, 1989.

[FDF00] E. Frazzoli, M.A. Dahleh and E. Feron. Real-time Motion Planning for Agile Autonomous Vehicles. *Proc. 2000 AIAA Conf. on Guidance, Navigation and Control*, Paper AIAA 2000-4056, Denver, CO, August 2000.

[GLM96] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. *Proc. ACM SIGGRAPH'96*, pp. 171-180, 1996.

[GHK99] L. Guibas, C. Holleman, and L. Kavraki. A Probabilistic Roadmap Planner for Flexible Objects with a Workspace Medial-Axis Based Sampling Approach. *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, 1999.

[HA01] L. Han and N.M. Amato. A Kinematics-Based Probabilistic Roadmap Method for Closed Chain Systems. In *Algorithmic and Computational Robotics: New Directions*, B.R. Donald, K.M. Lynch, and D. Rus (eds.), A K Peters, Natick, MA, pp. 233-246, 2001.

[HCK+00] K.E. Hoff III, T. Culver, J. Keyser, M. Lin, and D. Manocha. Interactive Motion Planning Using Hardware-Accelerated Computation of Generalized Voronoi Diagrams. *Proc. IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, 2000.

[HKW98] C. Holleman, L. Kavraki, and J. Warren. Planning Paths for a Flexible Surface Patch. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 21-26, 1998.

[HST94] T. Horsch, F. Schwarz, and H. Tolle. Motion Planning for Many Degrees of Freedom – Random Reflections at C-Space Obstacles. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 3318-3323, 1994.

[Hsu00] D. Hsu. *Randomized Single-Query Motion Planning in Expansive Spaces*. PhD Thesis, Computer Science Dept., Stanford University, Stanford, CA, May 2000.

[HKL+98] D. Hsu, L. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin. On Finding Narrow Passages with Probabilistic Roadmap Planners. In P.K. Agarwal et al. (eds.), *Robotics: The Algorithmic Perspective*, A K Peters, Natick, MA, pp. 151-153, 1998.

[HLM97] D. Hsu, J.C. Latombe and R. Motwani. Path Planning in Expansive Configuration Spaces. *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 2719-2726, 1997. A revised version of this paper can be found in *Int. J. of Computational Geometry and Applications*, 9(4-5):495-512, 1999.

[HLM+99] D. Hsu, J.C. Latombe, R. Motwani, and L.E. Kavraki. Capturing the Connectivity of High-Dimensional Geometric Spaces by Parallelizable Random Sampling Techniques. In *Advances in Randomized Parallel Computing*, P.M. Pardalos and S. Rajasekaran (eds.), Combinatorial Optimization Series, Kluwer, Boston, MA, pp. 159-182, 1999.

[HLS99] D. Hsu, J.C. Latombe, and S. Sorkin. Placing a Robot Manipulator Amid Obstacles for Optimized Execution. *Proc. IEEE Int. Symp. on Assembly and Task Planning (ISATP'99)*, Porto, Portugal, pp. 280-285, 1999.

[HXCW98] Y.K. Hwang, P.G. Xavier, P.C. Chen, and P.A. Wattergerg. Motion Planning with SANDROS and the Configuration Space Toolkit. In *Practical Motion Planning in Robotics*, K.K. Gupta and A.P. del Pobil (eds.), pp. 55-77, John Wiley & Sons, 1998.

[Hub95] P.M. Hubbard. Collision *Detection for Interactive Graphics Applications*. PhD Thesis, Computer Science Dept., Brown University, Providence, RI, 1995.

[JX01] X. Ji and J. Xiao. Planning Motion Compliant to Complex Contact States. *Proc. 2001 IEEE Int. Conf. on Robotics and Automation*, Seoul, Korea, May 2001.

[KZ86] K.G. Kant and S.W. Zucker. Toward Efficient Trajectory Planning: Path Velocity Decomposition. *Int. J. of Robotics Research*, 5:72-89, 1986.

[Kav94] L.E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD Thesis, Computer Science Dept., Stanford University, Stanford, CA, December 1994.

[KKL98] L.E. Kavraki, M. Kolountzakis, and J.C. Latombe. Analysis of Probabilistic Roadmaps for Path Planning. *IEEE Tr. on Robotics and Automation*, 14(1):166-171, Feb.1998.

[KSLO96] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces, *IEEE Tr. on Robotics and Automation*, 12(4):566-580, 1996.

[Kin01] R. Kindel. Motion Planning *for Free-Flying Robots in Dynamic and Uncertain Environments*. PhD Thesis, Aeronautics & Astronautics Dept., Stanford University, Stanford, CA, October 2001.

[KHLR00] R. Kindel, D. Hsu, J.C. Latombe, S. Rock. Kinodynamic Motion Planning Amidst Moving Obstacles. *Proc. IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, pp. 537-543, April 2000.

[KHM98] J. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-dops. *IEEE Trans. On Visualization and Computer Graphics*, 4(1): 21-37, 1998.

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

[KL94] Y. Koga and J.C. Latombe. On Multi-Arm Manipulation Planning. *Proc. IEEE Int. Conf on Robotics and Automation*, San Diego, CA, May 1994, pp. 945-952.

[KPL+98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries. In P.K. Agarwal et al. (eds.), *Robotics: The Algorithmic Perspective*, A K Peters, Natick, MA, pp. 177-190, 1998.

[Kuff99] J.J. Kuffner Jr. *Autonomous Agents for Real-Time Animation*, PhD Thesis, Computer Science Dept., Stanford University, Stanford, CA, December 1999.

[KNK+01] J.J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots under obstacle and dynamic balance constraints. *Proc. IEEE 2001 Int. Conf. on Robotics and Automation*, Seoul, Korea, May 2001.

[LK02] A. Ladd and L.E. Kavraki. A Measure Theoretic Analysis of PRM. *Manuscript*, 2002.

[LamK99] F. Lamiraux and L. Kavraki. Path Planning for Elastic Plates under Manipulation Constraints. *Proc. IEEE Int. Conf. on Robotics and Automation*, Detroit, MI, 1999.

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

[LS01] J.P. Laumond and T. Siméon. Notes on Visibility Roadmaps and Path Planning. In *Algorithmic and Computational Robotics: New Directions*, B.R. Donald, K.M. Lynch, and D. Rus (eds.), A K Peters, Natick, MA, pp. 317-328, 2001.

[LK99] S.M. LaValle and J.J. Kuffner. Randomized Kinodynamic Planning. *Proc. IEEE Int. Conf. on Robotics and Automation*, Detroit, MI, pp. 473-479, 1999.

[LK01] S.M. LaValle and J.J. Kuffner. Randomized Kinodynamic Planning. *Int. J. of Robotics Research*, 20(5):278-300, May 2001.

[LYK99] S.M. LaValle, J. Yakey, and L. Kavraki. A Probabilistic Roadmap Approach for Systems with Closed Kinematic Chains. *Proc. IEEE Int. Conf. on Robotics and Automation*, Detroit, MI, pp. 151-156, 1999.

[LLS99] S. Leroy, J.P. Laumond, and T. Siméon. Multiple Path Coordination for Mobile Robots: A Geometric Algorithm. *Proc. 16th Int. Joint Conf. on Artificial Intelligence*, Stockholm, Sweden, 1999.

[Lin00] M.C. Lin. Fast and Accurate Collision Detection for Virtual Environments. *Proc.IEEE Scientific Visualization Conf.*, 2000.

[LC91] M.C. Lin and J. Canny. A Fast Algorithm for Incremental Distance Calculation. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1008-1014, 1991.

[Mir98] B. Mirtich. V-Clip: Fast and Robust Polyhedral Collision Detection. *ACM Tr. on Graphics*, 17(3):177-208, 1998.

[NK00] C. Nielsen and L. Kavraki. A Two-Level Fuzzy PRM for Manipulation Planning. Proc. *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2000

[NSL99] C. Nissoud, T. Simeon, and J.P. Laumond. Visibility Based Probabilistic Roadmaps. *Proc. IEEE Int. Conf. on Intelligent Robors and Systems*, Detroit, MI, pp. 1316-1321, 1999.

[OLP89] P.A. O'Donnell and T. Lozano-Perez. Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators. *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 484-489, 1989.

[Ove92] M. Overmars. *A Random Approach to Motion Planning*. Tech. Rep. RUU-CS-92-32. Computer Science Dept., TB Utrecht, The Netherlands, 1992.

[PG95] L.J. Palmer and R.L. Grimsdale. Collision Detection for Animation Using Sphere-Trees. *Computer Graphics Forum*, 14(2):105-116, 1995.

[PSI92] A.P. del Pobil, M.A. Serna, and J. Iovet. A New Representation for Collision Avoidance and Detection. *Proc. IEEE Int. Conf. On Robotics and Automation*, 246-251, 1992.

[Qui94] S. Quinlan. Efficient Distance Computation Between Non-Convex Objects. *Proc. IEEE Int. Conf. On Robotics and Automation*, pp. 3324-3329, 1994.

[San02] G. Sánchez-Ante. *Single-Query Bi-Directional Motion Planning with Lazy Collision Checking*. PhD Thesis, ITESM, Campus Cuernavaca, Mexico, 2002.

[SL01] G. Sánchez-Ante and J.C. Latombe. Single-Query Bi-Directional Motion Planning with Lazy Collision Checking. *Proc. 10th Int. Symp. On Robotics Research*, Lorne, Australia, Nov. 2001.

[SL02] G. Sánchez-Ante and J.C. Latombe. Using a PRM Planner to Compare Centralized and Decoupled Planning for Multi-Robot Systems. *Proc. IEEE Int. Conf. on Robotics and Automation*, Washington D.C., May 2002.

[SS83] J.T. Schwartz and M. Sharir. In the Piano Mover's Problem: III. Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Moving Amidst Polygonal Barriers. *Int. J. of Robotics Research*, 2(3):46-75, 1983.

[SSLO98] S. Sekhavat, P. Švetska, J.P. Laumond, and M. Overmars. Multilevel Path Planning for Nonholonomic Robots Using Semiholonomic Subsystems. *Int. J. of Robotics Research*, 17:840-857, 1998.

[SLL01] T. Simeon, J.P. Laumond, and F. Lamiraux. Move3D: A Generic Platform for Motion Planning. *Proc. IEEE Int. Symp. on Assembly and Task Planning*, 2001.

[SLB99] A.P. Singh, J.C. Latombe, and D.L. Brutlag. A Motion Planning Approach to Flexible Ligand Binding. *Proc. 7th Int. Conf. on Intelligent Systems for Molecular Biology (ISMB)*, AAAI Press, Menlo Park, CA, pp. 252-261, 1999.

[SA01] G. Song and N.M. Amato. Using Motion Planning to Study Protein Folding Pathways. *Proc. 5th Int.l Conf. on Computational Molecular Biology (RECOMB'01)*, April 2001, pp. 287-296.

[SMA01] G. Song, S. Miller, and N. Amato. Customizing PRM Roadmaps at Query Time. *Proc. IEEE Int. Conf. On Robotics and Automation*, Seoul, Korea, May 2001.

[SR00] S.N. Spitz and A.A.G. Requicha. Multiple-Goals Path Planning for Coordinate Measuring Machines. *Proc. IEEE Int. Conf. On Robotics and Automation*, San Francisco, CA, April 2000.

[SHO93] A.F. van der Stappen, D. Halperin, and M.H. Overmars. The Complexity of the Free Space for a Robot Moving Amidst Fat Obstacles. *Computational Geometry: Theory and Applications*, 3:353-373, 1993.

[SRA01] S. Sundaram, I. Remmler, and N.M. Amato. Disassembly Sequencing Using a Motion Planning Approach. *Proc. IEEE Int. Conf. on Robotics and Automation,* May 2001, pp. 1475-1480.

[Sve97] P. Švetska. *Robot Motion Planning Using Probabilistic Roadmaps*. PhD Thesis, Utrecht University, 1997.

[SO95] P. Švetska and M. Overmars. Coordinated Motion Planning for Multiple Car-Like Robots Using Probabilistic Roadmaps. *Proc. IEEE Int. Conf. Robotics and Automation,* Nagoya, Japan, pp. 1631-1636, 1995.

[TAL99] R.Z. Tombropoulos, J.R. Adler, and J.C. Latombe. CARABEAMER : A Treatment Planner for a Robotic Radiosurgical System with General Kinematics. *Medical Image Analysis*, 3(3):237-264, 1999.

[Tou86] P. Tournassoud. A Strategy for Obstacle Avoidance and its Application to Multi-Robot Systems. *Proc. IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, pp. 1224-1229, 1986.

[VRA01] D. Vallejo, I. Remmler, N.M. Amato. An Adaptive Framework for `Single Shot' Motion Planning: A Self-Tuning System for Rigid and Articulated Robots. *Proc. IEEE Int. Conf. on Robotics and Automation*, Seoul, Korea, pp. 21-26, May 2001.

[War90] C.W. Warren. Multiple Robot Path Coordination Using Artificial Potential Fields, *Proc. IEEE Int. Conf. on Robotics and Automation,* Cincinnati, OH, pp. 500-505, May 1990.

[WAS99] S.A. Wilmarth, N.M. Amato, and P.F. Stiller. MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space. *Proc. IEEE Int. Conf. on Robotics and Automation* Detroit, MI, pp. 1024-1031, 1999.

[WHW99] C. Wuril, D. Henrich, and H. Worn. Multi-Goal Path Planning for Industrial Robots. *Proc. Int. Conf. Robotics and Application,* Santa Barbara, CA,, Oct. 1999.