A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking

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` and `latombe@cs.stanford.edu`

Abstract: This paper describes the foundations and algorithms of a new probabilistic roadmap (PRM) planner that is: *single-query* –it does not pre-compute a roadmap, but uses the two input query configurations to explore as little space as possible; *bi-directional* –it searches the robot's free space by concurrently building a roadmap made of two trees rooted at the query configurations; and applies a systematic *lazy collision-checking* strategy – it postpones collision tests along connections in the roadmap until they are absolutely needed. Several key observations motivated this collision-checking strategy: (1) PRM planners spend a huge fraction of their time (more than 90%) checking collision; (2) most connections in a PRM 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 a priori probability of testing collision-free. The strengths of single-query and bi-directional sampling techniques, and those of lazy 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, e.g., problems involving multiple robots in geometrically complex environments.

1. Introduction

Probabilistic roadmaps (PRM) have proven to be an effective tool to capture the connectivity of a robot's collision-free space and solve path-planning problems with many degrees of freedom (dofs) [Kav94, KSL+96, BKL+97, HLM+99] and/or complex admissibility constraints (e.g., nonholonomic, stability, dynamic, and visibility constraints) [KL00, HKL+01]. 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 explicitly compute 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 a PRM planner finds a collision-free path, if one exists, goes to 1 exponentially in the number of milestones [HLM97, Hsu00]. This property is called *probabilistic completeness*.

Not surprisingly, PRM planners spend most of their time performing collision checks (often much more than 90%). Several approaches are possible to reduce the total cost of collision checking:

- Design faster collision-checking algorithms. However, a number of efficient algorithms already exist [Lin00]. For example, *hierarchical decomposition* algorithms pre-compute a multi-level bounding approximation of every object in an environment (e.g., [Qui94, CLMP95, GLM96, KPL+98, KHM98]). At each collision query, they use the pre-computed bounding hierarchies to quickly discard large subsets of the objects that cannot possibly be involved in a collision. They scale up well to environments where object surfaces are described by several 100,000 triangles [HLM97].

- Design smarter sampling strategies yielding smaller roadmaps. For example, the strategy in [KSL+96] produces a first roadmap by sampling configuration space uniformly; next, it picks additional milestones in neighborhoods of existing milestones with no or few connections to the rest of the roadmap (as these milestones often lie in "difficult" regions of the free space). Other strategies generate a greater density of milestones near the boundary of the free space, since the connectivity of narrow regions is more difficult to capture (thus, requires more milestones) than that of wide-open regions [ABD98, HKL+98].

- Postpone collision-checking operations until they are absolutely needed (the so-called *lazy collision checking* strategy). This idea was introduced in [BK00]. The planner described in [BK00] distributes points uniformly at random in configuration space. It initially assumes that all points and connections between them are collision-free. It computes the shortest path in this network between two query configurations and tests the connections along this path for collision. If a collision is detected, the node and/or connection where it occurs are erased, and a new shortest path is computed and tested; and so on.

Lazy collision checking is a promising approach, but we think that its potential has only been partially exploited in [BK00]. The network built by this planner is reminiscent of a roadmap pre-computed by a multi-query planner [Kav94, KSL+96]. 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 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 in order to allay its limitations. This additional step is performed when the basic strategy fails to find a path; it adds milestones in regions selected around existing milestones. However, it is only an expedient, not a systematic solution.

The planner that we propose here exploits lazy collision checking more systematically, in particular by combining it with single-query, bi-directional sampling techniques similar to those introduced in [HLM97, Hsu00]. It 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 uses partial results of previous computation to locally adjust 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 seem to be cluttered with obstacles. It does not immediately test connections between milestones for collision. It is only when a sequence of milestones joining the two query configurations is created that the connections between milestones along this path are tested. This test is performed at successive points carefully ordered according to their likelihood of revealing a collision. 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 dual 1-GHz Pentium III processor, the planner 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 reduces the number of collision tests by a factor of 20, or more, in cluttered environments. The running time is reduced by roughly the same factor.

This increase in efficiency makes it possible to use the same planner to generate coordinated paths for multiple robots sharing the same environment. There are two established approaches to multi-robot motion planning [Lat91]: centralized and decoupled planning. Centralized planning has been seldom used in the past because it leads to searching configuration spaces with many dimensions. Instead, decoupled planning breaks the planning problem into smaller problems, but the approach is inherently incomplete. Interestingly, our planner can be used without change to perform either approach. We experimented with it on a spot-welding station with 6 robots combining 36 dofs in total. When performing centralized planning, our planner has been fully reliable and reasonably fast. When performing decoupled planning, it was slightly 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). Overall, our experimental results show that centralized planning is a more desirable approach.

This paper is organized as follows. Section 2 defines terms and notations used throughout the paper. Section 3 establishes the foundations of our planner, which is described in Section 4. Section 5 presents experimental results with the implemented planner on a single robot. Section 6 describes the application of the planner to coordinate multiple robots. The main contributions of this paper are the experimental evidence explaining why postponing collision checking is desirable, the lazy collision-checking strategy that is derived from this explanation, its incorporation into a single-query, bi-directional PRM planner, and the use of the same planner to coordinate multiple robots using centralized and decoupled planning approaches.

2. Definitions and Notations

Let C denote the configuration space of a robot 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. F is not explicitly represented. Instead, a *distance-computation procedure* is available. For any configuration $q \in C$, this procedure returns the Euclidean distance $D(q)$ (in the workspace) between the robot placed in configuration q and the obstacles, i.e., the distance between the two closest points in the robot and the obstacles. We get $D(q) > 0$ whenever $q \in F$ and $D(q) = 0$ otherwise.

We define a metric d over C. A simple and convenient one is the L_∞ metric. It is used in our implementation, but others would work as well. For any $q \in C$, the *neighborhood* of q of radius r is the subset $B(q,r) = \{q' \in C \mid d(q,q') < r\}$. With the L_∞ metric, this neighborhood is an n-D cube.

Consider a pair of configurations q and q'. When the robot moves from q to q' along a straight-line segment in C, every point on the robot traces a curve in the workspace. Let $L(q,q')$ be the Euclidean length of the longest such curve, over all robot's points. For any given robot, there exists a constant $\alpha > 0$ such that $L(q,q') < \alpha \times d(q,q')$; in the following, the tighter α the better. 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]. Hence, checking a straight segment connecting two configurations 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 the segment is collision-free; else break the segment into two halves and test each half recursively.

A PRM planner is given two *query configurations* as inputs, the "initial" configuration q_{init} and the "goal" configuration q_{goal}. If these configurations lie in the same connected component of F, the planner should return a collision-free path between them. There are two main classes of PRM planners: *multi-query* and *single-query* planners. A multi-query planner pre-computes a roadmap for a given robot and workspace, and then uses this roadmap to process multiple queries [Kav94, KSL+96]. In general, the query configurations are not known in advance, so the roadmap must be distributed over the entire free space; in particular, if the free space has multiple connected components, there should be a one-to-one map between these components and those of the pre-computed roadmap. Instead, a single-query planner computes a new roadmap for each query [HLM97, Kuf99, Hsu00]. Its only goal is to find a collision-free path between the two query configurations. The less free space it explores before finding such a path, the better. Multi-query planners are appropriate when the relatively high cost of pre-computing a roadmap can be amortized over a large number of queries. Single-query planners are more suitable when the number of queries is small and in environments with frequent changes (e.g., when obstacles are added, removed, or displaced between queries). Intermediate planners, which only pre-compute a partial roadmap, have also been proposed to handle small numbers of queries in the same environment [SMA01].

A single-query planner can take advantage of the fact that it knows the query configurations to only explore restricted subsets of the free-space components reachable from the query configurations. This can be done either by growing one tree of milestones from q_{init} or from q_{goal}, until a connection is found with the other query configuration (*single-directional* search), or by growing two trees concurrently, respectively rooted at q_{init} and q_{goal}, until a connection is found between the two trees (*bi-directional* search) [Hsu00]. In both cases, milestones are iteratively added to the roadmap. Each new milestone *m'* is selected in a neighborhood of a milestone *m* already installed in a tree and is connected to *m* by a local path (hence, *m'* becomes a child of *m*). Bi-directional planners are usually more efficient than single-directional ones. But other types of search are possible. For example, in addition to growing trees of milestones rooted at the query configurations, a single-query PRM planner could also grow disconnected islands of milestones, each island being initialized by picking a milestone at random in *F*. The final path would then be obtained by connecting the two trees through one or several successive islands. However, we are not aware of any implemented planner using such strategy.

The planner presented in this paper is a single-query, bi-directional PRM planner. Unlike previous such planners, it does not immediately test the connections between milestones for collision; instead, it applies a lazy-collision-checking strategy. Therefore, rather than referring to the connection between two adjacent nodes in a roadmap tree as a *local path*, we call it a *segment*. We first present the experimental data on which the collision-checking strategy is based; next we describe the new planner.

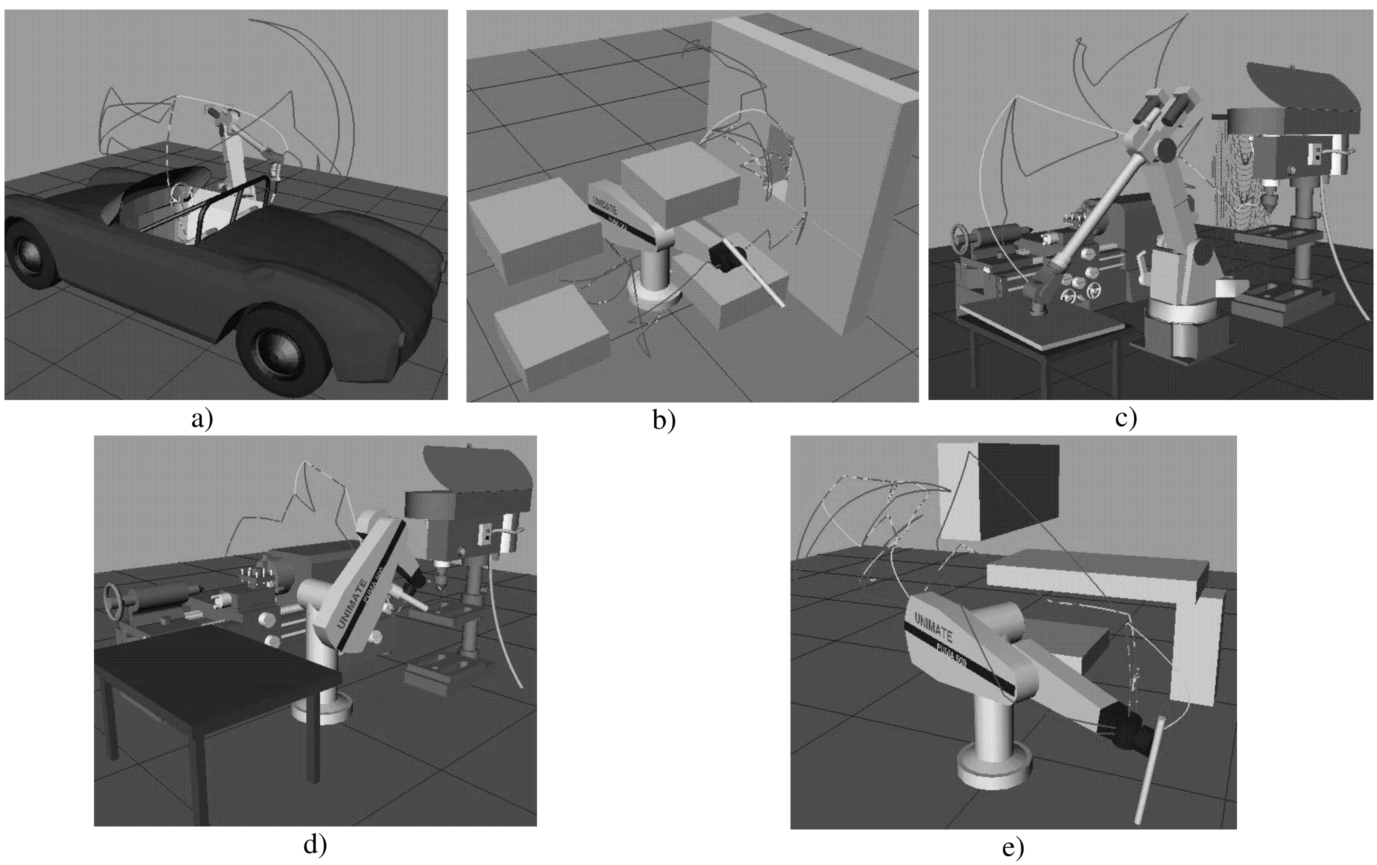
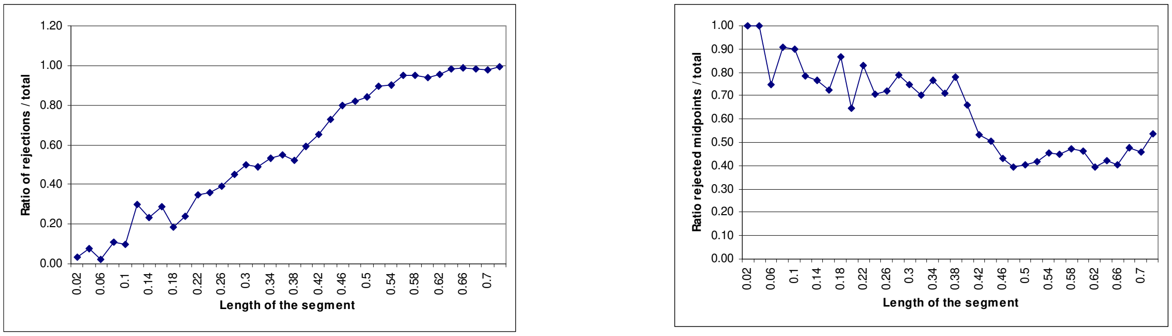


Figure 1: Path planning environments

3. Experimental Foundations

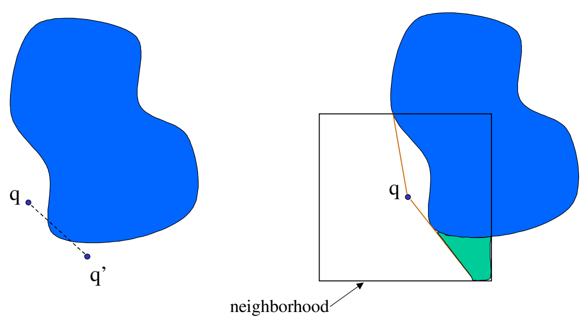
The design of our planner was suggested by experiments that we performed with the single-query PRM planner described in [Hsu00]. To study the impact of collision checking on the running time, we modified the planner's code by removing collision checks for connections between milestones (hence, only the 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 1 shows environments in which we made this observation.



a) b)

Figure 2: Collision ratios along connections

Every connection created by the planner between two milestones 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 and close to each other, then the straight-line segment between them has high a priori probability of being collision-free. To verify this analysis, 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 decomposed the range [0,1] of possible segment lengths into 50 equal-sized intervals and we used rejection sampling to eventually get the same number of segments in each interval. We then tested each of the 10,000 segments for collision. Figure 2a (generated for the environment of Figure 1a) displays the ratio of the number of segments that tested collision-free in each interval. Here, a segment shorter than 0.3 has probability greater than 0.6 of being collision-free. Similar charts were obtained with the other environments. In addition, we found out that, if a short connection is colliding, its midpoint has high probability to collide. Figure 2b shows, for each interval of Figure 2a, the fraction of colliding segments whose midpoints are colliding.



a) b)

Figure 3: Illustration for the result of Figure 2

There is a simple explanation for the results of Figure 2. Since the robot and the obstacles are "thick" in all or most directions, the obstacle regions in C are also thick (or fat [SHO93]) in most directions. 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. Indeed, consider Figure 3, 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 3a). Assume that q has been picked first (necessarily near the free space's boundary) and that q' is selected next inside the rectangular neighborhood $B(q,\rho)$ of q of L_∞ radius ρ (the threshold distance below which two configurations are considered close to each other). The gray (green) subset of this neighborhood (shown in Figure 3b) 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. Note that 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 thin walls.

This result and other experimental tests led us to make the following key observations:

- Most local paths in a PRM are not on the final path. Using the planner of [Hsu00] in the environments of Figure 1, we measured that the ratio of milestones on the final path varies between 0.01 and 0.001.

- The test of a local path 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.

- The chart of Figure 2a reveals that a short connection between two milestones has high probability of being collision-free. Thus, testing PRM connections early is likely to be both useless and expensive.

- The chart of Figure 2b shows that if a connection 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)

The lazy collision-checking strategy embedded in our planner and described in Section 4 derives directly from these observations. The planner described in that section assumes that the obstacle regions in C are thick in most directions, a condition satisfied in most practical problems.

4. Description of the Planner

The planner is given two parameters: s – the maximum number of milestones that the planner 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.

4.2. Overall algorithm

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

The planner 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-TREE adds a milestone to one of the two trees, while CONNECT-TREES tries to connect the two trees. The planner returns *failure* if it has not found a solution path after s iterations at Step 2 (a timeout condition can be used as well). If the planner returns *failure*, either no collision-free path exists between q_{init} and q_{goal}, or the planner failed to find one.

4.3. Tree expansion

Algorithm EXPAND-TREE
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 $i = 1, 2, \ldots, k$ 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 forced alternation between the two trees (Step 1) is aimed at avoiding that one eventually grows much bigger than the other, since we would then lose the advantages of bi-directional search. The number $\eta(m)$ associated with each milestone m in the selected tree T measures the current density of milestones of T around m. (Implementation details will be given in Subsection 4.6.) A milestone m is picked from T with probability inverse of $\eta(m)$ and a collision-free configuration q is picked at close distance (less than ρ) from m. This configuration q is the new milestone. The use of the probability distribution $\pi(m) \sim 1/\eta(m)$ 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 subset(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. An alternative technique for choosing m, which is proposed in [KL00], consists of first picking a configuration q in C uniformly at random and then selecting m to be the milestone in T that is the closest to q. However, if C contains large obstacle regions (which is often the case in practice), this technique biases the choice of the milestone toward those regions, which may be undesirable.

Step 2.2 selects a series of up to k 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.

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 outweighed by the computational gain brought by the postponement of collision checking.

4.4. Tree connection

Algorithm CONNECT-TREES
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-TREE. Let m' be the closest milestone to m in the other tree. If these two milestones are less than ρ apart, then CONNECT-TREES 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.

4.5. 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 $\alpha \times d(q,q') \le \max\{D(q),D(q')\}$ and *maybe* otherwise.

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 $D(q_{\mathrm{mid}}) = 0$ 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 $u_1, u_2, \ldots, u_p$ 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.

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.

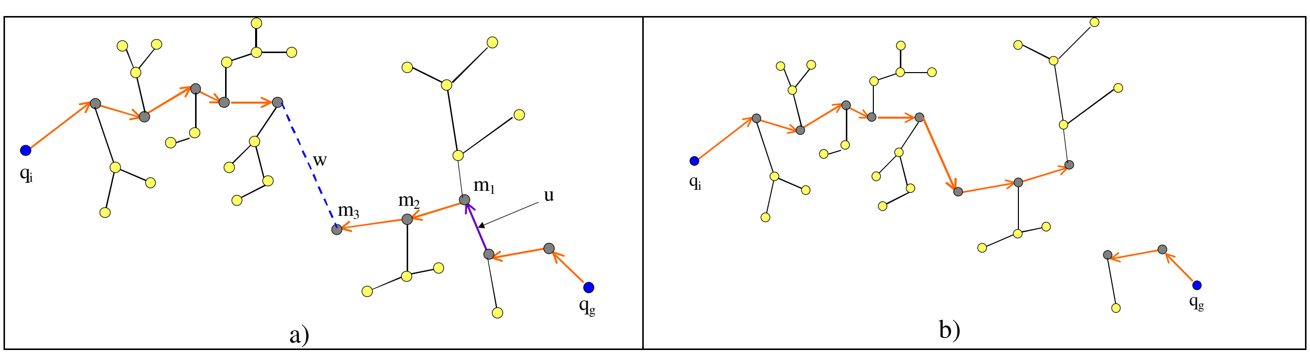


Figure 4: Transfer of milestones from one tree to the other

The removal of a segment u disconnects again the roadmap into two trees. If u is the bridge that CONNECT-TREES created to connect the two trees, the two trees return to their previous state (except for the *Unsafe* sets of some segments, whose contents may have changed). Otherwise, the removal of u results in a transfer of milestones from one tree to the other. Assume that u is in T_{goal}, as illustrated in Figure 4a, where $w \neq u$ denotes the bridge added by CONNECT-TREES. The milestones $m_1, \ldots, m_r$ between u and w (r = 3 in Figure 4) and their children in T_{goal} are transferred to T_{init} as shown in Figure 4b. The parent-child connections between the transferred milestones remain the same, except those between $m_1, \ldots, m_r$, which are inverted. In this way, no milestone is ever removed from the roadmap. The collision-checking work done along all segments, except the one that tested to collide, is saved in the *Unsafe* sets. Hence, if one of the segments later lies on another candidate path, the distance computations previously done are not repeated.

4.6. Implementation details

Collision checking. We have implemented the planner with two collision checkers, PQP [GLM96] and Quinlan's [Qui94]. Each rigid object in an environment is described by a collection of triangles representing its surface. The checker pre-computes a bounding hierarchical representation of each object – oriented-bounding boxes for PQP and spheres for Quinlan's checker. No other pre-computation is done.

Both PQP and Quinlan's algorithms can be used to compute exact or approximate distances, or to only check whether a collision occurs. In our planner's implementation, we use them as pure collision checkers, that is, for any configuration q they return whether the robot collides with obstacles, or not. To check that 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*.

The planner associates a collision-check index $\kappa(u)$ with each segment u between milestones (including the bridge). This index takes an integer value indicating the resolution at which u has already been tested. If $\kappa(u) = 0$, then only the two endpoints of u (which are both milestones) have been tested collision-free. If $\kappa(u) = 1$, then the two endpoints and the midpoint of u have been tested collision-free. More generally, for any $\kappa(u)$, $2^{\kappa(u)}+1$ equally distant points of u have been tested collision-free. Let $\lambda(u)$ be the length of u. If $2^{-\kappa(u)}\,\lambda(u) < \varepsilon$, then u is marked *safe*. The index of every 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 $q \in \sigma(u,j+1) \backslash \sigma(u,j)$
   - 2.1. If $D(q) = 0$, then return *collision*
3. If $2^{-(j+1)}\lambda(u) < \varepsilon$
   - 3.1. Then mark u as *safe*
   - 3.2. Else $\kappa(u) \leftarrow j+1$

For every segment u that is not marked *safe*, the current 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 Step 1 of TEST-PATH results in increasing the index of the segment u that is in first position in U by 1.

Spatial Indexing. The planner 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-TREE, where we pick a milestone m from one tree T with a probability distribution $\pi(m) \sim 1/\eta(m)$. Rather than maintaining the density $\eta(m)$ around each milestone, which could be quite costly, 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-computing the arrays A_{init} and A_{goal}, but the total cost of this operation is negligible relative to collision checking.

The two arrays A_{init} and A_{goal} are also used at Step 2 of CONNECT-TREES to identify the milestone m' that will be connected to the newly added milestone m. The implemented CONNECT-TREES attempts two connections, instead of one. First, 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. Note that m and m' are then only guaranteed to be close to each other along h dimensions. The second attempt picks m' uniformly at random in the other tree. Our experiments have shown that on average this technique is faster than connecting m to the closest milestone. (The "closest-milestone" heuristics tends to postpone the finding of some easy connections.)

Path optimization. Finally, we added a simple path optimizer to our planner. 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 connecting them tests collision-free, replace the portion of τ between q and q' by this segment. In this way, blatant jerks are eliminated.

5. Experimental Results

The planner is written in C++ and compiled with g++ 2.5.3. The running times reported below were gathered on a dual 1-GHz Pentium III processor with 1 GB of main memory running Linux. The planner uses the PQP checker also written in C++. The distance threshold ρ was set to 0.15 and the collision-checking resolution ε to 0.01. Each of the indexing arrays A_{init} and A_{goal} had size 10×10. The two dimensions of C indexed in these arrays are selected uniformly at random among all the dimensions of C, and changed whenever 50 new milestones have been generated. The pre-computation time of PQP is not included in the running times given below.

5.1. Environments

The environments in Figure 1 are among those we used to test our planner. 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 1a 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 1b is another 6-dof arm. The environment was designed to produce narrow passages in configuration space. Such passages (e.g., the passage letting the bar go through 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 1c, the robot must transfer a large metal sheet from a table to a press. n_{rob} = 5,000; n_{obs} = 83,000.

- In Figure 1d, 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 1e shows another environment designed to create narrow passages. n_{rob} = 3,000; n_{obs} = 50.

The planner computes robot paths in configuration space. In Figure 1, we only display 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.

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 5: Experimental results in the environment of Figure 1a

5.2. Basic performance evaluation

Figure 5 shows results for the environment of Figure 1a 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 1a corresponds to the third line in the table of Figure 5. In that example, the optimized path took an additional 0.19 seconds to generate.

Example	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
1a	0.60	159	13	1483	342	245	0.58	0.38
1b	4.45	1609	39	11211	411	7832	4.21	2.48
1c	4.42	1405	24	7267	277	3769	4.17	1.86
1d	0.17	33	10	406	124	47	0.17	0.07
1e	6.99	4160	44	12228	447	6990	6.30	3.55

Figure 6: Statistics on experiments in the environments of Figure 1

The table in Figure 6 shows averages for 100 runs of the planner on five examples, each in a different environment of Figure 1. The last column indicates the standard deviation on the running times. In all cases, the planner found a path in reasonable time; 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 collision 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.

5.3. 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 between two milestones before inserting it in the roadmap. This planner is similar to those presented in [Hsu00, KL00]. 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 the lazy planner considers any collision-free configuration q picked in the neighborhood of a milestone m as a new milestone (Step 2.2 of EXPAND-TREE), the second 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(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
1a	2.82	22	5	7425	173	83	2.81	3.01
1b	106.20	3388	32	300060	421	9504	105.56	59.30
1c	18.46	771	16	38975	219	3793	18.35	15.34
1d	1.03	29	9	2440	123	46	1.02	0.70
1e	293.77	6737	24	666084	300	11971	292.40	122.75

Figure 7: Experimental results for the full-collision check planner

Figure 7 shows experimental 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 less than for the full-collision-check planner, especially in problems b, c, and e, that have the longest running times. Moreover, in example 1b, 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. Unfortunately, these results cannot be compared to those in [BK00], where the improvement was measured relative to a traditional *multi-query* planner. Unlike a single-query planner, such a planner pre-computes a large roadmap to cover the entire free space.

We also ran the two planners on the same query configurations in four versions of the environment of Figure 1e obtained by changing the size of some obstacles, hence the widths of narrow passages in free space. In the easiest version, the lazy planner did 963 collision checks and took 0.41 seconds, with the other planner did 7,238 checks and took 6.14 seconds (average number over 10 runs). In a more difficult version of the environment, the numbers were 6,172 checks and 8.98 seconds for the lazy planner, vs. 84,190 checks and 71.64 seconds for the other planner. This result indicates that, though lazy collision checking does not directly address the "narrow passage" issue analyzed in [HKL98], the relative efficiency of the lazy planner increases as passages get narrower.

6. Multi-Robot Planning

In this section we describe the use of our planner to compute coordinated paths for multiple robots sharing a subset of their respective workspaces. Spot-welding stations in automotive body shops (see Figure 8), are a good example of such multi-robot systems, with a number of robot dofs typically ranging between 20 and 50.

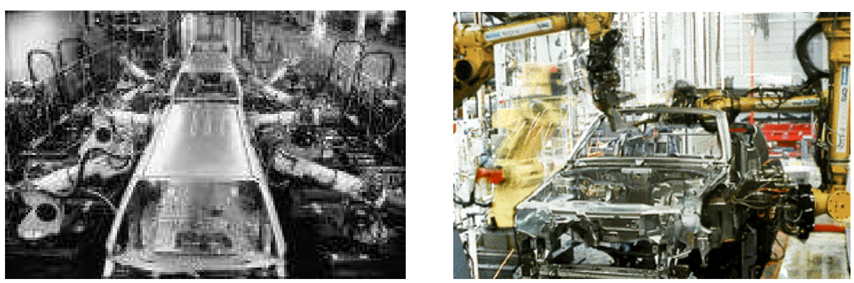


Figure 8: Spot-welding stations in automotive body shops

6.1. Centralized vs. decoupled planning

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

- *Centralized planning* consists of considering the robots as if they were forming a single multi-limb robot, by encoding all their dofs in a single configuration space and searching for a collision-free path in that space. The number of dimensions of the configuration space is equal to the total number of dofs of the robots. A collision-free path in this space, if one exists, not only describes the individual path to be followed by each robot, but also how the robots are to be coordinated. In the past, centralized planning was not considered a practical approach because it leads to searching configuration spaces with too many dimensions. Most proposed techniques are based on ad-hoc and incomplete heuristics, for example potential field techniques, which are too unreliable 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].

- *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 [KZ86, OLP89, AIQ98, AdB+99]. Decoupled planning leads to searching lower-dimensional spaces than centralized planning. 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 [ELP86]. 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.

6.2. Velocity tuning

Velocity tuning consists of searching a "reduced" configuration space. To illustrate, consider two robots whose respective paths, τ_1 and τ_2, have been planned independently. 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 = 1,2) 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. 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 [OLP86]; it sets the relative velocities of the robots along their respective paths. Note that this path may not be monotonic along any of the dimensions of P. If it is not 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 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, one may coordinate all the robots by generating a collision-free path in the p-dimensional 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 spaces $P_2,\ldots,P_p$. The axes of P_2 encode the lengths along the paths of the 1st and 2nd robots, and 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 path 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 spaces.

6.3. Implemented multi-robot planners

We have developed three multi-robot path planners respectively based on centralized planning, decoupled planning with global coordination, and decoupled planning with pairwise coordination. Interestingly, they all use the planner of Section 4.

The first planner is essentially the same as in Section 4.

The second planner makes p+1 calls to the planner of Section 4, where p is the number of robots: p calls to plan the path of each robot (ignoring the other robots); then, another call to plan a collision-free path in the space $P = [0,L_1] \times \ldots \times [0,L_p]$. The roadmap in P is built as described in Section 4 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 rigid bodies belonging to two distinct robots.

The third planner makes $2p$-1 calls to the planner of Section 4: p calls to plan the path of each robot; then, p-1 calls to plan collision-free paths in the 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 rigid bodies belonging to two robots among the robots 1 to i.

All three planners use the path optimizer described in Subsection 4.6. In the second and third planners, this optimizer is called after the robot paths have been coordinated.

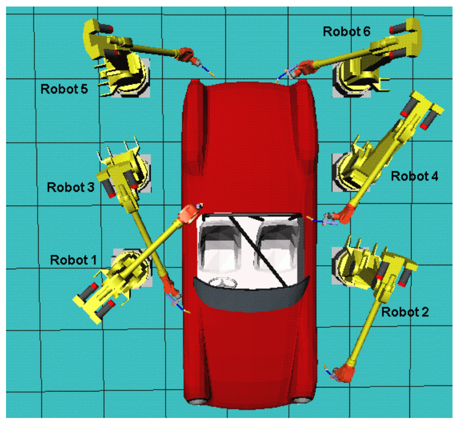


Figure 9: Six-robot environment

6.4. Experimental results

We have conducted experiments with the three planners in the environment of Figure 9, which represents a spot-welding station similar to those in Figure 8. This station contains six robots with six dofs each. In order to keep our planners as general possible, we did not take advantage of certain properties of this environment. For example, the planners assume that collisions may occur between any two bodies of any two robots, while it is clear that many pairs of bodies cannot collide. So, since each robot consists of 6 rigid bodies, testing that no two robots collide at a given configuration requires running the collision checker on 540 pairs of bodies. Fortunately, when two bodies are sufficiently far apart, the pre-computed bounding hierarchies lead the checker to quickly detect that they cannot possibly collide.

In general, if p is the number of robots and b the number of bodies in each robot, the checker would be called $b^2p(p\text{-}1)/2$ times; therefore, for values of p and b larger than those in Figure 9, the brute-force enumeration of every pair of bodies would no longer be acceptable. We may then exploit the prior knowledge (specific to the environment of Figure 9) that only a small number of pairs of bodies can possibly collide. Alternatively, general spatial indexing techniques [Aga97] may be used to select the pairs of bodies that should be tested at each collision check.

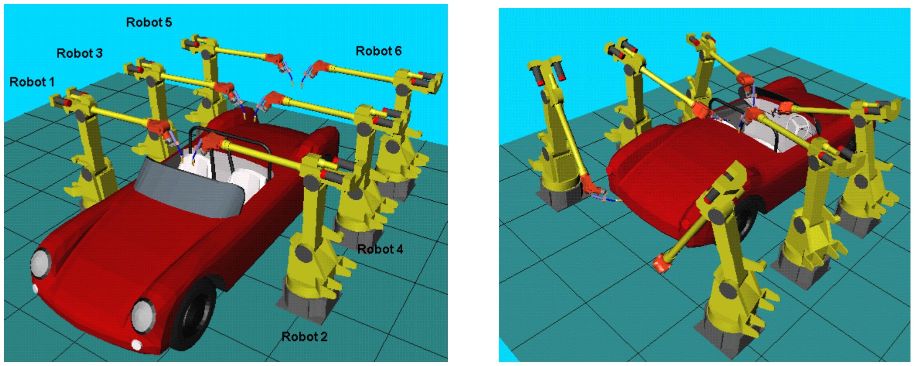


Figure 10: Problem I

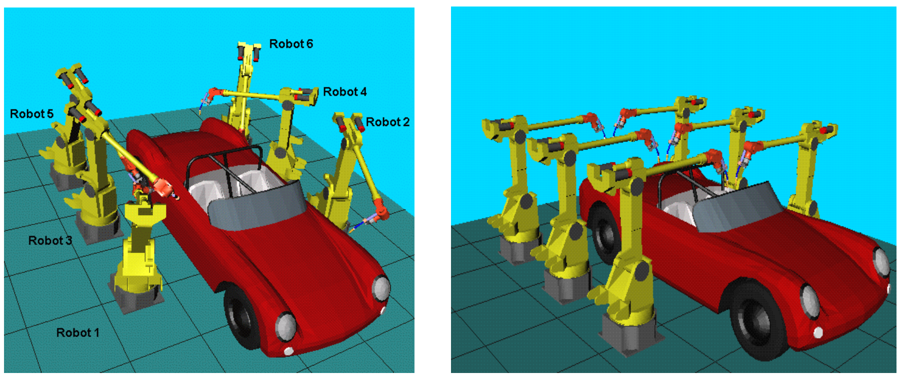


Figure 11: Problem II

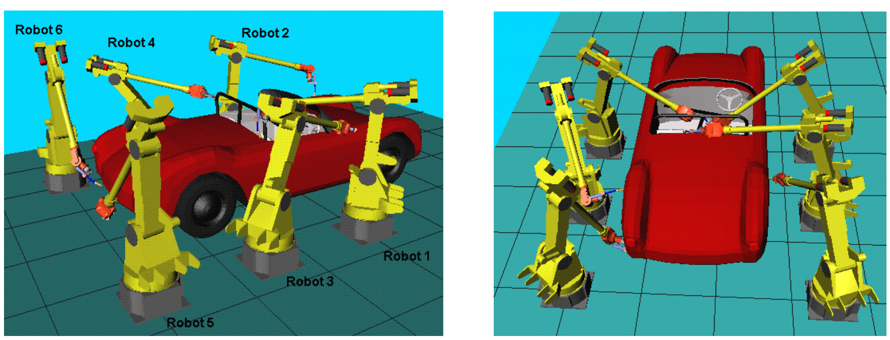


Figure 12: Problem III

Figures 10, 11, and 12 show the initial and goal configurations of the robots for three problems (I, II, and III) on which we ran our planners. The table of Figure 13 shows statistics (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 as 1, 2, …, 6, as shown in Figure 9. A problem for 2 (or 4) robots is defined as shown in Figures 10, 11, and 12, 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 14 lists the largest and smallest running times and number of milestones over the 100 runs for each of the 3×3 problems. Figure 15 shows a series of snapshots along a path computed for problem III with two robots.

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 13: 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 14: Largest/smallest running time and number of milestones for centralized planner

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 bodies 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 pairs of bodies tested. The reason is that each collision check involves testing each robot body against the environment and testing every pair of bodies from two different robots against each other. On average the second type of test is less costly than the first, on the one hand because the model of the environment contains more triangles than that of a robot body, and on the other hand because many two robot bodies 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.

The table of Figure 16 lists the number of failures and the average running times of our three planners over 20 runs on each of problems I, II, and III, with 2, 4, and 6 robots. The average running times are computed over the runs when the planner successfully returns a path. A failure was returned when velocity tuning in a given space failed after generating 50,000 milestones. The table of Figure 17 shows the average number of milestones in the roadmap when a planner succeeds in finding a solution. The relatively small numbers of milestones indicate that a failure of a decoupled planner is caused by the incompleteness of the approach, not by the probabilistic completeness of our planner. As we expected, the results in Figure 16 show that the decoupled planner with global coordination is significantly less reliable than the centralized planner, but more reliable than the decoupled planner with pairwise coordination. The running times are not easily comparable. For the more complex Problem III with 4 and 6 robots, decoupled planning seems faster than centralized planning, but it also fails often.

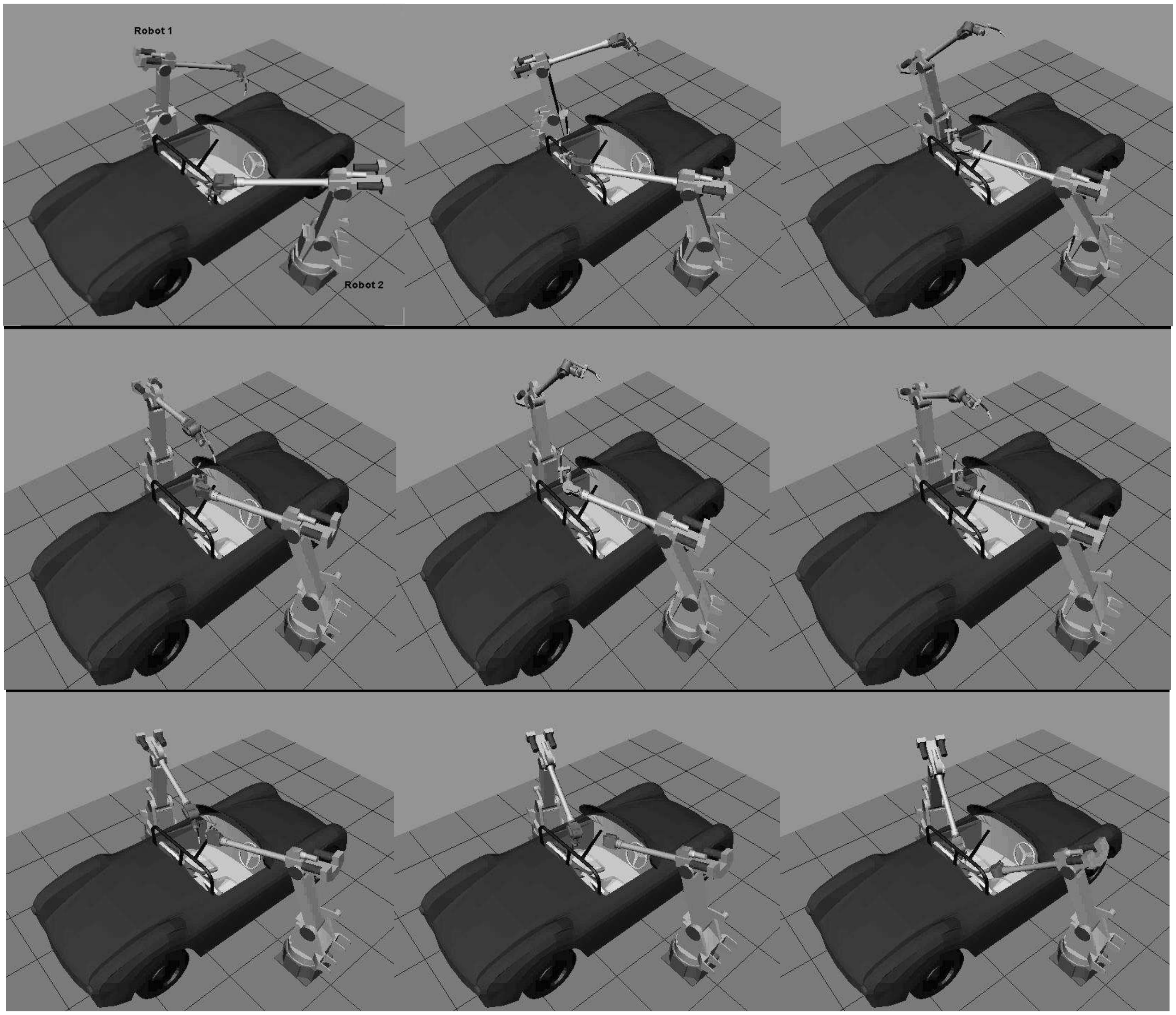


Figure 15: Snapshots along a path computed by the centralized planner for problem III with two robots

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 16: Comparison of centralized and decoupled planners

	2 Robots			*4 Robots*			*6 Robots*		
	Prob. I	**Prob. II**	**Prob. III**	**Prob. I**	**Prob. II**	**Prob. III**	**Prob. I**	**Prob. II**	**Prob. III**
Centralized	11	11	191	62	56	842	322	882	5648
Dec. Global	8	10	254	34	43	652	423	1546	2651
Dec. Pairwise	10	6	278	23	30	234	254	342	1321

Figure 17: Average numbers of milestones in the roadmaps

7. Conclusion

The planner's design and the experimental results presented in this paper demonstrate that a PRM planner combining a lazy collision-checking strategy with single-query bi-directional sampling techniques 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 a single or few robots, and from seconds to a few minutes for multiple robots. 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 or Quinlan's algorithms for computing distances rather than as a pure collision checkers; this could significantly reduce the number of calls to this program. As mentioned earlier, in multi-robot systems, we may speed-up collision checking by using general space indexing techniques and environment-specific simplifications to reduce the number of pairs of bodies tested.

Our main current goal, however, is to extend the planner to facilitate the programming of multi-robot spot-welding stations in automotive body shops (Figure 8). A typical such station include 4 to10 robots, each sharing a portion of its workspace with 1 to 4 other robots. The geometric model of a station (robots and car bodies) consists 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 (e.g., 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 robot trajectories as quickly as possible. Some extensions of the planner will require additional research. For example, 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 related problems can be found in [BSB99, SR00, WHW99].

Acknowledgements: The research reported in this paper was conducted in the Computer Science Department at Stanford University. It was funded in part by ARO MURI grant DAAH04-96-1-007 and by grants from General Motors Research and ABB. G. Sánchez's stay at Stanford was supported in part by 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 PQP collision checker was made available to us by S. Gottschalk, M. Lin, and D. Manocha from the Computer Science Department at the University of North-Carolina in Chapel Hill.

References

[Aga87] 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.

[AIQ98] R. Alami, F. Ingrand, and S. Qutub. A scheme for coordinating multi-robot planning activities and plans execution. *Proc. 13th European Conf. on Artificial Intelligence* (*ECAI98*), pp. 617-621, 1998.

[ABD98] 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.

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

[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.

[BK00] R. Bohlin and L.E. Kavraki Path Planning Using Lazy PRM. *Proc. IEEE Int. Conf. Robotics & Autom.*, 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.

[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.

[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.

[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 & Autom.*, 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

[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.

[KSL+96] 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 & Autom.* 12(4):566-580, 1996.

[HKL+01] D. Hsu, R. Kindel, D. Hsu, J.C. Latombe, and S. Rock. Randomized Kinodynamic Motion Planning with Moving Obstacles. In *Algorithmic and Computational Robotic: New Directions*, B.R. Donald, K.M. Lynch, and D. Rus (eds.), A K Peters, Natick, MA, pp. 247-264, 2001.

[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.

[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.

[KL00] J.J. Kuffner and S.M. LaValle. RRT-Connect: An Efficient Approach to Single-Query Path Planning. *Proc. IEEE Int. Conf. Robotics and Autom.,* San Francisco, CA, April 2000.

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

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

[OLP86] P.A. O'Donnell and T. Lozano-Perez. Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators. *Proc. IEEE Int. Conf. Robotics & Autom.*, pp. 484-489, 1989.

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

[SS93] 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.

[SMA01] G. Song, S. Miller, and N. Amato. Customizing PRM Roadmaps at Query Time. *Proc. IEEE Int. Conf. On Robotics & Autom.*, 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 & Autom.*, 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.

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

[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.