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

Gildardo Sánchez[1] **and Jean-Claude Latombe**[2]

[1]Computer Science Dept., ITESM Campus Cuernavaca, Cuernavaca, México, gsanchez@campus.gda.itesm.mx

[2]Computer Science Dept., Stanford University, Stanford, CA, USA, latombe@cs.stanford.edu

Abstract

This paper describes the foundations and algorithms of a new probabilistic roadmap (PRM) planner that is: (1) *single-query* –i.e., it does not pre-compute a roadmap, but uses the two input query configurations to explore as little space as possible; (2) *bi-directional* i.e., it searches the robot's free space by concurrently building a roadmap made of two trees rooted at the query configurations – and (3) applies a systematic *lazy collision-checking* strategy –i.e., it postpones collision tests along connections in the roadmap until they are absolutely needed. Several observations motivated this collision-checking strategy: (1) PRM planners spend more than 90% of their time checking collision; (2) most connections in a PRM are not on the final path; (3) the collision test for a connection is the most expensive when there is no collision; and (4) the probability that a short connection is collision-free is large. The strengths of single-query and bi-directional sampling techniques, and those of lazy collision checking reinforce each other. This combination reduces planning time by large factors, making it possible to handle more difficult planning problems, including multi-robot problems 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) [1, 2, 3] and/or complex admissibility (e.g., nonholonomic, stability, dynamic, and visibility constraints) [4, 5]. 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 it is often impractical to explicitly compute the collision-free subset (the *free space*) of a configuration space, but there exists collision-detection algorithms that efficiently checks whether a given configuration or local path is collision-free [1]. 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 [6, 7].

PRM planners spend most of their time performing collision checks (often much more than 90%). Several approaches are possible to reduce the overall cost of collision checking:

- Design faster collision-checking algorithms. However, efficient techniques already exist, e.g., some checkers pre-compute a hierarchy of bounding volumes for every object in an environment [8, 9, 10]. For each collision query, they then use the hierarchies to quickly rule out large portions of objects that cannot collide. The scale up well to environments where object surfaces are described by several 100,000 triangles [6].

- Design smarter sampling strategies. For example, the strategy in [3] produces a first roadmap by sampling the configuration space uniformly. Next, it picks additional milestones in neighborhoods of existing milestones with no or few connections to the rest of the roadmap. Other strategies generate a greater density of milestones near the boundary of the free space, as the connectivity of narrow regions is more difficult to capture than that of wide-open regions [11, 12].

- Postpone collision tests until they are absolutely needed (*lazy collision checking*). The planner in [13] 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 it for collision. If a collision is detected, the node and/or segment where it occurs are erased, and a new shortest path is computed and tested; and so on.

We think that lazy collision checking is a promising approach, but that its potential has only been partially exploited in [13]. The network built is reminiscent of a roadmap pre-computed by a multi-query planner [2, 3]. 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 for collision. The focus on shortest paths may also be inappropriate when obstacles force the robot to take long detours.

In this paper, we present a new PRM planner – called SBL, for *S*ingle-query, *B*i-directional, *L*azy collision checking – that tries to better exploit lazy collision checking, in particular by combining it with single-query, bi-directional sampling techniques similar to those in [6, 7]. 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 locally adjusts the sampling resolution, in order to take larger steps in wide-open regions of the free space and smaller steps in narrow regions. It does not immediately test connections between milestones for collision. Only when a sequence of milestones joining the two query configurations is found, the connections between milestones along this path are tested, and this test is performed at successive points ordered according to their likelihood of revealing a collision. So, 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, the planner reliably solves problems for 6-dof robots in times ranging from a small fraction of a second to a few seconds. Comparison with a similar planner using a traditional collision-checking strategy shows that the lazy strategy cuts the number of collision tests and the running time by a factor from 4 to 6 for moderately cluttered environments like those shown in Fig. 1 (a)-(b)-(c) to 20 to 40 for more cluttered environments like those shown in Fig. 1(d)-(e), to over 40 for more cluttered environments. SBL also solves multi-robot problems reliably and efficiently, like the one shown in Fig. 5 (36 dof in total).

Section 2 defines terms and notations used throughout this paper. Section 3 provides the foundations of the lazy collision-checking strategy. Section 4 describes the SBL algorithms. Section 5 presents experimental results on single-robot (6 dof) and multi-robot (up to 36 dofs) problems. The planner's code can be downloaded from http://robotics.stanford.edu/~latombe/projects/. Movies showing runs of the planner are also available.

2. Definitions and Notations

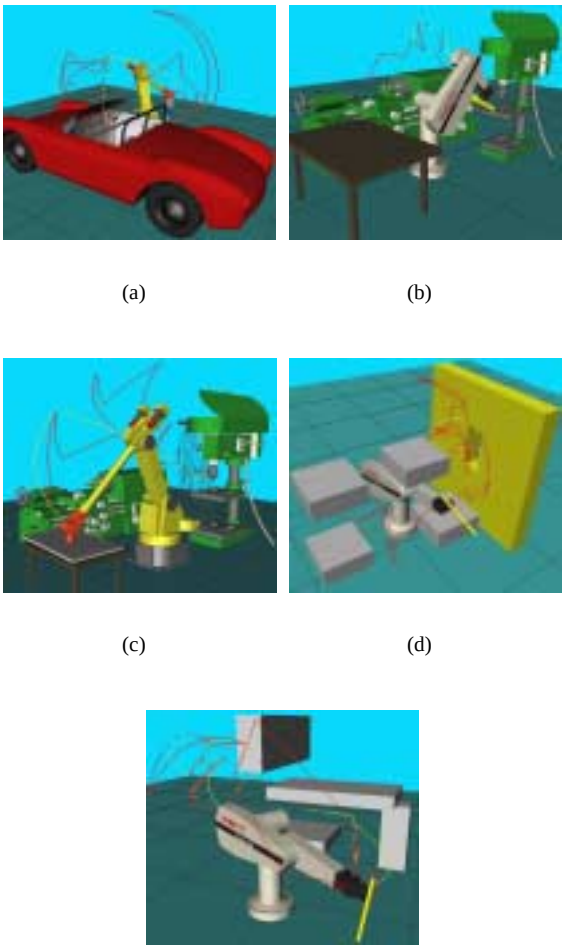
Let $\mathcal{C}$ denote the configuration space of a robot and $\mathcal{F} \subset \mathcal{C}$ its free space. We normalize the range of values of each dof to be $[0, 1]$ and we represent $\mathcal{C}$ as $[0, 1]^n$, where n is the number of dofs of the robot. We define a metric d over $\mathcal{C}$. Our implementation of SBL uses the simple L_∞ metric. For any $q \in \mathcal{C}$, the neighborhood of q of radius r is the subset $B(q, r) = \{q' \in \mathcal{C} | d(q, q') < r\}$. With the L_∞ metric, it is an n-D cube.

No explicit geometric representation of $\mathcal{F}$ is computed. Instead, given any $q \in \mathcal{C}$, a collision checker returns whether $q \in \mathcal{F}$. A path τ in $\mathcal{C}$ is considered collision-free if a series of points on τ, such that every two successive points are closer apart than some ε, are all collision-free. A rigorous test (eliminating the need for ε) is possible by using a distance-computation algorithm instead of a pure collision checker [1, 7].

A query to a PRM planner is defined by two *query configurations*, q_{init} and q_{goal}. If these configurations lie in the same component of $\mathcal{F}$, the planner should return a collision-free path between them; otherwise, it should indicate that no such path exists. There are two main classes of PRM planners: *multi-query* and *single-query*. A multi-query planner pre-computes a roadmap which it later uses to process multiple queries [2, 3]. To deal with any possible query, the roadmap must be distributed over the entire free space. Instead, a single-query planner computes a new roadmap for each query [6, 7, 14]. The less free space it explores to find a path between the two query configurations, the better. Single-query planners are more suitable in environments with frequent changes.

A single-query planner either grows one tree of milestones from q_{init} or q_{goal}, until a connection is found with the other query configuration (*single-directional* search), or grows two trees concurrently, respectively rooted at q_{init} and q_{goal} until a connection is found between the two trees (*bi-directional* search) [7]. 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 T, and is connected to m by a local path (hence, m' becomes a child of m in T). Bi-directional planners are usually more efficient than single-directional ones.

SBL is a single-query, bi-directional PRM planner. Unlike previous such planners, it does not immediately test the connections between milestones for collision. Therefore, rather than referring to the connection between two adjacent nodes in a roadmap tree as a *local path*, we call it a *segment*.



(a) (b)

(c) (d)

(e)

Figure 1: Path planning environments.

3. Experimental Foundations

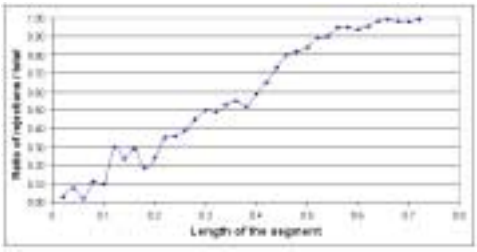
The design of our planner was suggested by experiments that we performed with the PRM planner described in [7]. To study the impact of collision-checking on the running time, we modified the planner's code by removing collision-checks for connections between milestones. As could be expected, the planner was faster by two to three orders of magnitude, but surprisingly a significant fraction of the generated paths were actually collision-free. Fig. 1 shows environments in which we made this observation.

Every segment created between two milestones by the planner of [7] is relatively short (less than 0.15). Thus, the above observation suggested that two configurations picked at random are both collision-free and close to each other, then the straight-line segment between them has high 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 (recall that the L_∞ diameter of $\mathcal{C}$ is 1). This was done by picking 100 collision-free configurations in $\mathcal{C}$ uniformly at random and connecting each such configuration q to 100 additional collision-free configurations obtained by randomly sampling neighborhoods of q of different diameters. 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. The charts of Fig. 2(a) and Fig. 2(b) display the ratio of the number of segments that tested collision-free in each interval in the environments of Fig. 1(a) and Fig. 1(e), respectively. In those cases, a segment shorter than 0.2 has probability greater than 0.7 of being collision-free. Similar charts were obtained with the other environments.

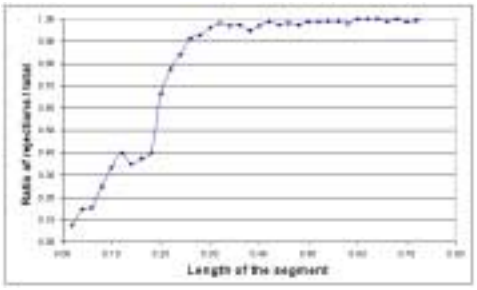
In fact, there is a simple explanation for the results of Fig. 2. Since the robot and the obstacles are "thick" in all or most directions, the obstacle regions in $\mathcal{C}$ are also thick (or fat [15]) in most directions. Hence, a short colliding segment with collision-free endpoints is necessarily almost tangential to an obstacle region in $\mathcal{C}$, an event that has small probability. Indeed, consider Fig. 3, where the dark 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 (Fig. 3(a)). Assume that q was picked first and that q' is selected next inside the rectangular neighborhood of q of a small L_∞ radius. In general, the subset of this neighborhood (shown in gray in Fig. 3(b)) in which q' must be selected in order to satisfy the above conditions is only a small fraction of the neighborhood's collision-free subset. The probability is small to first pick q close enough to the boundary of $\mathcal{F}$ and next pick q' in the appropriate subset of the neighborhood of q. Note that this would not be true if the obstacle regions in $\mathcal{C}$ were thin, e.g., if the robot was a point moving in a planar maze of thin walls.

We also observed that, if a short segment is colliding, then its midpoint has high probability to collide. Fig. 2(c) 2(d) gives, for each interval of Fig. 2(a) 2(b), the measured fraction of colliding segments whose midpoints are colliding.

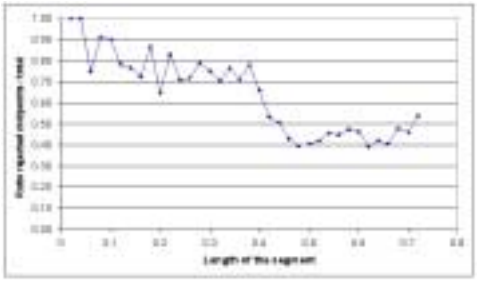
The above results and other tests led us to make the following key observations:



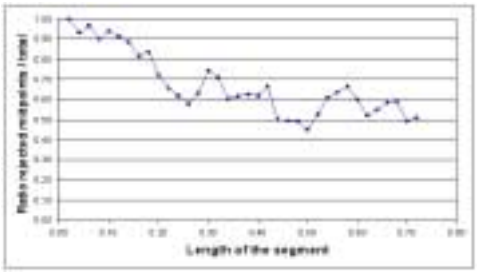
(a)



(b)



(c)



(d)

Figure 2: Collision ratios among connections.

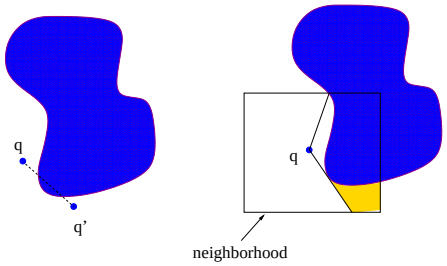


Figure 3: Illustration of the result of Figs. 2.

1. Most local paths in a PRM are not on the final path. Using the planner of [7] in the environments of Fig. 1, we measured that the ratio of milestones on the final path varies between 0.01 and 0.001.
2. 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.
3. A segment between two milestones has high probability of being collision-free.
4. If a segment is colliding, its midpoint has high probability to be in collision.

Observations 1, 2, and 3 indicate that testing segments early is likely to be both useless and expensive, while observation 4 suggests that segments should be tested by recursively breaking them into halves.

4. Description of the Planner

SBL 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.

4.1. 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
3       EXPAND-TREE
4       τ ← CONNECT-TREES
5       If τ ≠ nil then return τ
6   Return failure
```

The planner builds two milestone trees, T_{init} and T_{goal}, respectively rooted at 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. If the planner returns *failure*, either no collision-free path exists between q_{init} and q_{goal}, or the planner failed to find one.

4.2. Tree expansion

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
3 Pick a milestone m from T at random, with probability $\pi(m) \sim 1/\eta(m)$
4 For $i = 1, 2, ..., k$ until a new milestone q has been generated
5 Pick a configuration q uniformly at random from $B(m, \rho/i)$
6 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. A number $\eta(m)$ is associated with each milestone m in this tree, which measures the current density of milestones of T around m. A milestone m is picked from T with probability inverse of $\eta(m)$ and a collision-free configuration q is picked at a 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 3 was introduced in [6] to avoid over-sampling regions of $\mathcal{F}$. It guarantees that the distribution of milestones eventually diffuses through the subsets of $\mathcal{F}$ reachable from from q_{init} and q_{goal}. In [6, 7] this condition is required to prove that the planner will eventually find a path, if one exists. The alternation between the two trees prevents any tree from eventually growing much bigger than the other, as the advantages of bi-directional search would then be lost.

Step 4 applies an adaptive sampling strategy, by selecting 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 $\mathcal{F}$ than in narrow regions.

4.3. Tree connection

CONNECT-TREES()
1 $m \leftarrow$ most recently created milestone
2 $m' \leftarrow$ closest milestone to m in the tree not containing m
3 If $d(m, m') < \zeta$ then
4 Connect m and m' by a bridge w
5 $\tau \leftarrow$ path connecting q_{init} and q_{goal}
6 Return TEST-PATH (τ)
7 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. The two trees are connected by a segment, called a *bridge*, between m and m' if these two milestones are less than some ζ apart. 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.

4.4. Path testing

The path-testing strategy of SBL derives from the charts shown in Fig. 2. 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 $\lambda(u)$ denote 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 algorithm TEST-SEGMENT(u) increases $\kappa(u)$ by 1:

TEST-SEGMENT(u)
1 $j \leftarrow \kappa(u)$
2 For every $q \in \sigma(u, j+1) \backslash \sigma(u, j)$
3 if q is in collision, then return *collision*;
4 if $2^{-(j+1)}\lambda(u) < \varepsilon$
5 Then mark u as *safe*
6 else $\kappa(u) \leftarrow j + 1$

For every segment u that is not marked safe, the cur-

rent value of $2^{-\kappa(u)}\lambda(u)$ is cached in the data structure representing u. The smaller this value, the greater the probability that u is collision-free.

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 sorted in decreasing order of $2^{-\kappa(u)}\lambda(u)$ of all the segments $\{u_1, u_2, ..., u_p\}$ that are not marked *safe*.

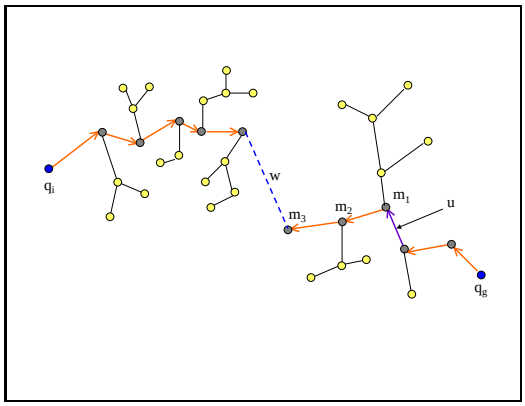
TEST-PATH(τ)
1 While U is not empty do
2 $u \leftarrow$ extract(U)
3 If TEST-SEGMENT(u) = *collision* then
4 Remove u from the roadmap
5 Return *nil*
6 If u is not marked *safe*, then re-insert it in U
7 Return τ

Each loop of Step 1 results in increasing the index on the segment u that is in first position in U. This segment is removed from U. It is re-inserted in U if TEST-SEGMENT(u) at Step 3 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 $2^{-\kappa(u)}\lambda(u)$ 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.

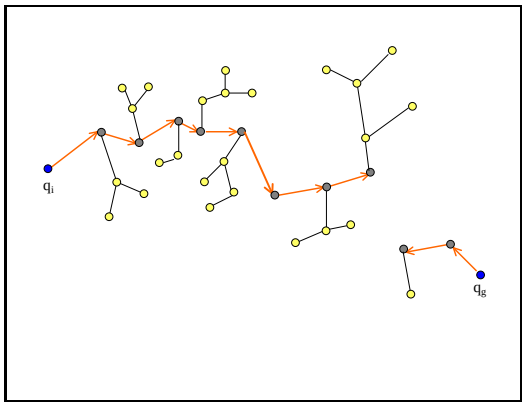
The removal of a segment u disconnects the roadmap into two trees. If u is the bridge that CONNECT-TREES created to connect the two trees, the two trees return to their previous state. Otherwise, the removal of u results in a transfer of milestones from one tree to the other. Assume that u is in T_{goal}, as illustrated in Fig. 4(a), where $w \neq u$ denotes the bridge added by CONNECT-TREES. The milestones $m_1, ..., m_r$ between u and w ($r = 3$ in Fig. 4(a)) and their children in T_{goal} are transferred to T_{init} as shown in Fig. 4(b). The parent-child connections between the transferred milestones remain the same, except those between $m_1, ..., m_r$, which are inverted. No milestone is removed from the roadmap. The collision-checking work done along all segments, except the one that tested to collide, is saved in their indices.

4.5. Implementation details

We have implemented SBL with several collision checkers, including PQP [9]. Each rigid object in an environment is described by a collection of triangles representing its surface. PQP pre-computes a bounding



(a)



(b)

Figure 4: Transferring milestones between trees.

hierarchy of oriented-bounding boxes for each object. No other pre-computation is done.

The planner spatially indexes every milestone of T_{init} (resp. T_{goal}) in an h-D array A_{init} (resp. A_{goal}). Both arrays partition the subspace defined by h dimensions of $\mathcal{C}$ (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 3 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, we first pick a non-empty cell of A_{init}, then a milestone from this cell. So, 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 $\mathcal{F}$ along the h selected dimensions. To ensure diffusion along all dimensions of $\mathcal{C}$, 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.

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. 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 along h dimensions. The second attempt picks m' uniformily at random in the other tree. Our experiments have shown that on average this technique is faster than just connecting m to the closest milestone. (The "closest-milestone" heuristic tends to postpone the finding of some easy connections.)

Finally, we added a simple path optimizer to eliminate blatant jerks. It takes a path τ as input and performs the following operation several (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.

5. Experimental Results

SBL is written in C++. The running times reported below were obtained on an 1-GHz Pentium III processor and 1Gb of main memory running Linux. The planner uses the PQP checker. The distance threshold ρ was set to 0.15 and the resolution ε was set to 0.01. Each of the indexing arrays A_{init} and A_{goal} had size 10×10. The two dimensions of $\mathcal{C}$ indexed in these arrays are selected uniformily at random among all the dimensions of $\mathcal{C}$, and changed whenever 50 new milestones have been generated. The pre-computation time of PQP is not included in the running times given below.

Table 1: Number of triangles in robot and obstacles.

	1a	1b	1c	1d	1e
n_{rob}	5,000	3,000	5,000	3,000	3,000
n_{obst}	21,000	50,000	83,000	100	50

Fig. 1 displays some of the single-robot examples we used to test our planer. In each example, the dark curve is traced by the center-point of the robot's end-effector for one non-optimized path generated by the planner. The light curve is defined in the same way for the optimized path. The numbers of triangles in the geometric models of the robot and the obstacles, n_{rob} and n_{obst}, in each example, are listed in Table 1.

The geometrically simpler examples 1d and 1e are intended to test SBL when the free space contains narrow passages, a notorious difficulty for PRM planners [12].

5.1. Basic performance evaluation

Table 2 gives averages over 100 runs of SBL on each of the five examples of Fig. 1. 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 large fraction of the collision checks were made on the solution path. As noticed in [13], these collision tests cannot be avoided. The running times in Table 2 do not include path optimization, which in all cases takes an additional 0.1 to 0.2s.

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 3-D arrays, but performance results were not significantly different.

5.2. 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 [4, 7]. 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 SBL considers any collision-free configuration q picked in the neighborhood of a milestone m as a new milestone (Step 4 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.

Table 3 shows results with the modified planner, on the same five problems. The maximal number of milestones s was set to 10,000 and the results are averages over 100 runs. The average running times of SBL are smaller than those of the full collision-checking planner by factors ranging from 4.4 for the problem in Fig. 1(c), to over 40 for the problem in Fig. 1(e).

Table 2: Statistics on the examples of Fig. 1.

	Total time (s)	Milest. in roadmap	Milest. in path	Total Nr. of Coll. Checks	Coll. Checks on the path	Samples tested	Time for Coll.Checks	Std. Dev. (s)
a	0.60	159	13	1483	342	145	0.58	0.38
b	0.17	33	10	406	124	47	0.17	0.07
c	4.42	1405	24	7267	277	3769	4.17	1.86
d	4.45	1609	39	11211	411	7832	4.21	2.48
e	6.99	4160	44	12228	447	6990	6.30	3.55

Table 3: Experimental results for the full-collision check planner.

	Total time (s)	Milest. in roadmap	Milest. in path	Total Nr. of Coll. Checks	Coll. Checks on the path	Samples tested	Time for Coll.Checks	Std. Dev. (s)
a	2.82	22	5	7425	173	83	2.81	3.01
b	1.03	29	9	2440	123	46	1.02	0.70
c	18.46	771	16	38975	219	3793	18.35	15.34
d	106.20	3388	32	300060	421	9504	105.56	59.30
e	293.77	6737	24	666084	300	11971	292.40	122.75

5.3. Multi-robot examples

We ran SBL on several problems in the environment of Fig. 5, which represents a spot-welding station found in automotive body shops. This station contains 6 robots each with 6 dofs each. SBL treats them as if they formed a single robot with 36 dofs, hence builds a roadmap in a 36-D configuration space. To keep SBL general, we did not take advantage of specific properties of the environment. For example, SBL assumes that collisions may occur between any two bodies of any two robots, while many pairs of bodies cannot collide.

Table 4 gives averages over 100 runs of SBL on 9 problems. Fig. 5 shows the initial and final configurations for the problem named PIII-6 in Table 4. PIII-2 and PIII-4 are the same problem, but reduced to robots 1 and 2, and robots 1 through 4, respectively. The problems PI-2/4/6 and PII-2/4/6 are simpler, with either the initial or the goal configurations being the rest configuration of the robots.

In all $100 \times 3 \times 3 = 900$ runs, SBL successfully returned a path in a satisfactory amount of time. In each run, the maximum number of milestones allowed to the planner was set to 10,000, but the maximum number of milestones actually generated by SBL was 6917 for a run of problem PIII-6. The increase in the running times when the number of robots goes from 2 to 4 to 6 is caused both by the quadratic growth in the number of pairs of bodies that are tested at each collision-checking operation and by the greater difficulty of the problems due to the constraints imposed

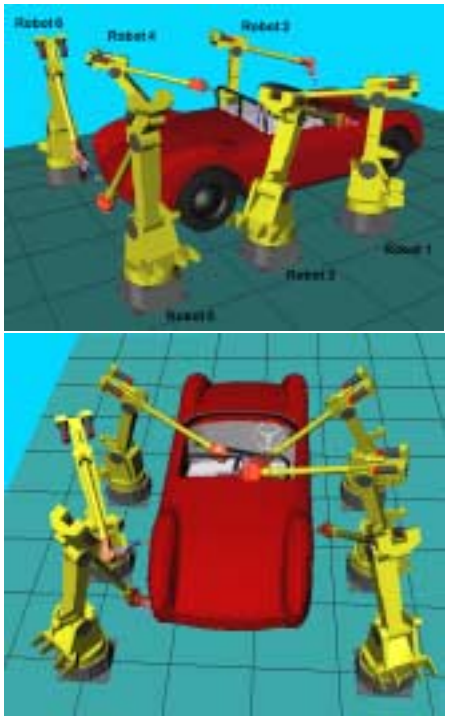


Figure 5: Multi-robot problem.

Table 4: Statistics on 9 multi-robot problems.

	Total time (s)	Milest. in roadmap	Milest. in path	Total Nr. of Coll. Checks	Coll. Checks on the path	Samples tested	Time for Coll.Checks	Std. Dev. (s)
PI-2 Robs	0.26	11	4	242	58	18	0.26	0.52
PII-2 Robs	0.25	11	5	248	76	13	0.25	0.17
PIII-2 Robs	2.44	191	17	2356	243	718	2.41	1.57
PI-4 Robs	3.97	62	7	1015	106	193	3.96	5.67
PII-4 Robs	3.94	56	10	968	166	112	3.93	2.4
PIII-4 Robs	30.82	841	32	8895	542	2945	3057	15.55
PI-6 Robs	28.91	322	14	3599	121	1083	28.82	28.91
PII-6 Robs	59.65	882	30	6891	533	1981	59.41	31.08
PIII-6 Robs	442.85	5648	91	47384	1525	24511	439.39	170.46

by the additional robots upon the motions of the other robots. We created a specific version of SBL that tests only the pairs of bodies that can possibly collide. For the most complex problem (PIII-6), the average running time was reduced to 323s.

6. Conclusion

This paper shows 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 example, we could use PQP for computing distances rather than as a pure collision checker. This could significantly reduce the number of calls to this program [1].

Our main goal is now to extend SBL to facilitate the programming of multi-robot spot-welding stations in automotive body shops. In particular, each robot must perform several welding operations, but the ordering of these operations is not fully specified. Hence, the planner will have to compute an optimized tour of the welding locations to be visited by each robot. This is a variant of the Traveling Salesman Problem, where the distance between two locations is not given and, instead, must be computed by SBL. Clearly, if there are r locations to visit, we do not want to invoke this planner $O(r^2)$ times; a better method must be found.

Acknowledgments

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. PQP was made available to us by S. Gottschalk, M. Lin, and D. Manocha from the Computer Science Dept. at the University of North-Carolina in Chapel Hill.

References

[1] 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.

[2] L. E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford University, 1994.

[3] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12(4):566–580, 1996.

[4] J.J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. Rob. & Autom.*, 2000.

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

[6] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Rob. & Autom.*, pages 2719–2726, 1997.

[7] D. Hsu. *Randomized Single-Query Motion Planning in Expansive Spaces*. PhD thesis, Stanford University, 2000.

[8] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-Collide: An interactive and exact collision detection system for large scale environments. In *Proc. ACM Interactive 3D Graphics Conf.*, pages 189–196, 1995.

[9] S. Gottschalk and M. Lin. OOBTree: A hierarchical structure for rapid interference detection. In *Proc. SIGGRAPH 96*, pages 171–180, 1996.

[10] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. IEEE Int. Conf. Rob. & Autom.*, pages 3324–3329, 1994.

[11] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In P. K. Agarwal et al., editor, *Robotics: The Algorithmic Perspective*, pages 155–168, 1998.

[12] D. Hsu, L. E. Kavraki, J. C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P. K. Agarwal et al., editor, *Robotics: The Algorithmic Perspective*, pages 151–153, Natick, MA, 1998.

[13] R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. In *Proc. IEEE Int. Conf. Rob. & Autom.*, pages 521–528, 2000.

[14] J. J. Kuffner. *Autonomous Agents for Real-Time Animation*. PhD thesis, Stanford University, 1999.

[15] 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.