

# Multiple Query Probabilistic Roadmap Planning using Single Query Planning Primitives

Kostas E. Bekris, Brian Y. Chen, Andrew M. Ladd, Erion Plaku, and Lydia E. Kavraki

*Department of Computer Science  
Rice University  
Houston TX, 77005*

{ bekris,brianyc,aladd,plakue,kavraki }@cs.rice.edu

## Abstract

We propose the combination of techniques that solve multiple queries for motion planning problems with single query planners. Our implementation uses a probabilistic roadmap method PRM with bidirectional rapidly exploring random trees BI-RRT as the local planner. With small modifications to the standard algorithms, we obtain a multiple query planner which is significantly faster and more reliable than its component parts. Our method provides a smooth spectrum between the PRM and BI-RRT techniques and obtains the advantages of both. We observed that the performance differences are most notable in planning instances with several rigid non-convex robots in a scene with narrow passages. This planner is in the spirit of non-uniform sampling and refinement techniques used in earlier work on PRM.

## 1 Introduction

Multiple query motion planning is motivated by applications where the robot operates in the same environment for a sufficiently long period of time. In these cases, a data structure is build in a preprocessing phase in order for many queries to be answered quickly. For applications where the environment is changing or the robot only sees a local window, a motion planner has to efficiently explore the space in order to solve a single query without preprocessing information. In this paper we use a single query motion planning algorithm as a subroutine for a multiple query planner. By combining these two approaches, a planner which uses the strengths of both is obtained.

The probabilistic roadmap method (PRM) approach to motion planning is efficient, easily implemented and applicable to a large variety of motion planning instances [15, 9]. It is a multiple query planner which constructs a roadmap by sampling points

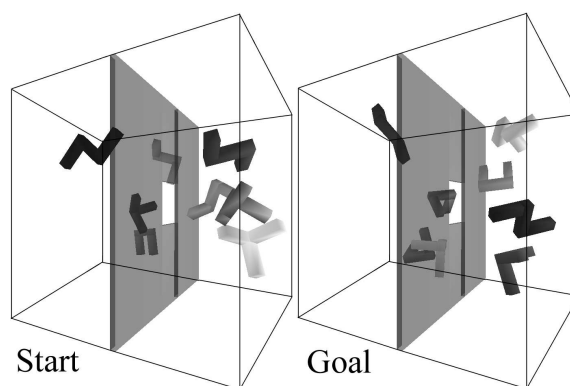


Figure 1: Scene for narrow

in the space and connecting them with a primitive planner. Many implementations use a variation of the straight line planner as the primitive local planner, although the algorithm is generic and the local planner only need satisfy a simple property to allow for a probabilistically complete planner [16]. At query time, a sufficiently good roadmap captures the structure and connectivity of the configuration space (C-space) well enough to resolve the query quickly.

Single query planning can be achieved with potential fields [5, 12] and more recently has been solved by growing trees in the C-space [11, 18]. Using a potential field planner as a local planner for PRM has been attempted [13]. There are also approaches that incrementally construct a roadmap by merging RRTs that are produced as queries are resolved [21].

In this work, we examine the applicability of rapidly exploring random trees (RRT) [18] as a local planner for a PRM. We are primarily interested in obtaining a planner which is effective for high-dimensional problems [17] as those arising in computational biology [4, 2]. Our implementation focuses on the multiple mover problem in three dimensions (Figure 1) by

modifying the originally stated RRT and PRM formulations so as to efficiently explore the C-space. We have tested our planner on a variety of benchmarks and we noticed significant improvement on clock time and roadmap quality compared to the PRM or BI-RRT planners. We report the results of our experiments and discuss how to vary the input parameters to obtain optimum performance.

## 2 Method and Implementation

**Heterogenous Two-Tiered Planning** The main contribution of this paper is to propose the combination of PRM and RRT methods in order to solve multiple query planning problems more efficiently. The general principle behind this work is to increase the power of the local planner in order to reduce the number of milestones needed by the PRM and to obtain a more robust planner.

We use BI-RRT as a local planner for a PRM planner. RRTs are grown at each milestone and connections between milestones are computed by BI-RRT. The RRTs rooted at the milestones and the paths between them are stored for use in queries. The use of RRT is akin to a refinement phase used in some PRM implementations [15] and the extra power of BI-RRT allows for fewer milestones in the top-level roadmap. Our implementation uses no specific refinement or non-uniform sampling heuristics and follows standard implementations from the literature.

**Configuration Space** The class of problems we considered for the purposes of our experiments with this general planning framework consisted of multiple non-convex, rigid bodies moving freely in a three dimensional workspace with rigid, non-convex static obstacles. As with other motion planners, obvious generalizations to many other kinematic planning problems are straightforward and the use of RRTs as subroutine gives a natural way to extend the planner to kinodynamic planning instances [7, 10, 19].

In our implementation, we represent the configuration of a single robot by a point and a quaternion  $(p, q)$ .  $N$  robots operating in the workspace are represented by a tuple  $\{(p_1, q_1), \dots, (p_N, q_N)\}$ . The dimensionality of the configuration space in this case is  $6N$ . To obtain an embedding of a single robot into Euclidean 6-D, we concatenate two points from opposite corners of the bounding box of a given configuration of the robot in the workspace. This construction is used for each robot to obtain a point in Euclidean  $6N$ -D. Distance is measured in the usual way for these points. A straight line between two configurations  $(s_i, s_j) = (p_i, q_i, p_j, q_j)$  is defined as a linear interpolation between the two translation points  $(p_i, p_j)$  and a spherical interpolation for the quater-

nion representations of the rotation  $(q_i, q_j)$ .

Our implementation uses SWIFT++ [8] for collision detection. Equally spaced points along the straight line between two configurations are tested for collision with intersection checks. The order of the checks is done by bisection, which increases the chances of quick rejection of paths in collision.

**PRM Implementation** We use a general implementation of PRM [15]. Our interpretation of the algorithm follows.

---

### Algorithm 1 BUILD ROADMAP( $n, k$ )

---

- 1: Generate a set of  $n$  configurations in free space  $S$  from some distribution.
  - 2: Let  $G = \emptyset$ , the empty graph on  $S$ .
  - 3: **for** each configuration  $s_i \in S$  **do**
  - 4:   Find  $k$  neighbors for  $s_i$ ,  $N_k(s_i)$ .
  - 5:   **for** each configuration  $s_j \in N_k(s_i)$  **do**
  - 6:     **if**  $j > i$  and the local planner can find a collision-free path from  $s_i$  to  $s_j$  **then**
  - 7:       add an edge  $(i, j)$  in  $G$ .
  - 8:       annotate  $(i, j)$  with the cost of the path.
  - 9:     **end if**
  - 10:   **end for**
  - 11: **end for**
  - 12: **return** the graph  $G$  and the set  $S$ .
- 

The  $k$  neighbor queries are answered using a combination of  $k$ -nearest neighbors and a random selection. The use of random selection offsets problems with the metric we observed in narrow areas of the space. As the graph is being built, the number of connected components can be maintained by using a fast union-find data structure. By limiting the number of intra-component local planner checks by a small constant, roadmaps can be constructed more quickly without sacrificing much quality. This pass is similar to approaches used in other PRM implementations [14].

**RRT Implementation** The top-level implementation of the RRT algorithm [20] is as follows:

---

### Algorithm 2 RRT( $n, s_0$ )

---

- 1: Add  $s_0$  as the root of the tree,  $T$ .
  - 2: **for**  $i$  ranges from 1 to  $n$  **do**
  - 3:   Generate a free configuration from a random distribution,  $s_{\text{target}}$ .
  - 4:   Find the closest point in the tree  $T$ ,  $s_j$ .
  - 5:   Set  $s_i$  to INCREMENTAL-PLANNER( $s_j, s_{\text{target}}$ ).
  - 6:   Add  $s_i$  to the tree,  $T$ , as a child of  $s_j$ .
  - 7:   Annotate the edge in the tree with the cost from  $s_j$  to  $s_i$ .
  - 8: **end for**
-

We use the following implementation of BI – RRT.

---

**Algorithm 3** BI-RRT( $n, T_1, T_2$ )

---

- 1: **for**  $i$  ranges from 1 to  $n$  **do**
  - 2:   Generate a free configuration from a random distribution,  $s_{\text{target}}$ .
  - 3:   Find the closest points in each tree,  $s_j^1$  and  $s_k^2$  respectively.
  - 4:   **if** the local planner can connect  $s_j^1$  to  $s_k^2$  **then**
  - 5:     **return** the path from  $s_0^1$  to  $s_0^2$  via  $s_j^1$  and  $s_k^2$ .
  - 6:   **end if**
  - 7:   Set  $s_i^1$  to INCREMENTAL-PLANNER( $s_j^1, s_{\text{target}}$ ).
  - 8:   **if** the local planner can connect  $s_i^1$  to  $s_k^2$  **then**
  - 9:     **return** the path from  $s_0^1$  to  $s_0^2$  via  $s_i^1$  and  $s_k^2$ .
  - 10:  **end if**
  - 11:  Swap  $T_1$  and  $T_2$ .
  - 12: **end for**
  - 13: **return** no path was found.
- 

**Adapting RRT for use with PRM** Each milestone in the PRM graph is an RRT. When generating the milestones, we begin with a random configuration for the root and grow the RRT by a fixed number of iterations. Each RRT can be viewed as a set of configurations, the embeddings we describe earlier are used to obtain a set of points in  $6N$ -D. The centroid of this point set is computed and is used as the coordinates for that RRT. In Algorithm 1, the neighbor query uses these coordinates.

The local planner for the PRM is Algorithm 3. Before running the BI-RRT, we first compute  $k$  close pairs between the two sets of configurations and try to use a straight line planner to connect them. In Algorithm 3, the local planner is taken to be a straight line planner.

In the case of multi-robot motion planning the generation of the node configurations is faster if it is done incrementally. We first choose a random order to generate the robots in and then embed the robots in this order. If the embedding fails at any point, we re-embed the robot that failed until it succeeds. This leads to a significant reduction in the number of attempts required.

The incremental planner that we use checks collisions by bisection and has been adapted for multi-robot planning instances. Each robot is moved simultaneously towards the goal configuration. The path is checked for collisions incrementally by adding one robot at a time to the path and checking for collisions with the environment and with the previous robots. If a collision is found, then a new target configuration for the robot being added is generated. Although this local planner is more expensive than checking all robots simultaneously, we found that it

is considerably more effective in covering the space. The configuration returned by the call is the final goal that computed. In the case where no robot can move, no configuration is returned and outer loop repeats without incrementing.

**$k$ -nearest neighbors** The  $k$ -nearest neighbor queries were implemented using  $6N$ -D tree [6]. In our experience, the efficiency of this query in practice is significantly better than the worst case  $O(n^{\frac{6N-1}{6N}})$  if some care is taken during the query. The tree is traversed greedily towards the query point. When walking up the tree, if the worst point in the partial  $k$ -NN is further than the closest point of the incrementally constructed bounding box around the point set on the other branch, then that branch is taken. Also, a small performance improvement can be gained by not splitting point sets of cardinality smaller than some threshold, experimentally around 25 to 80.

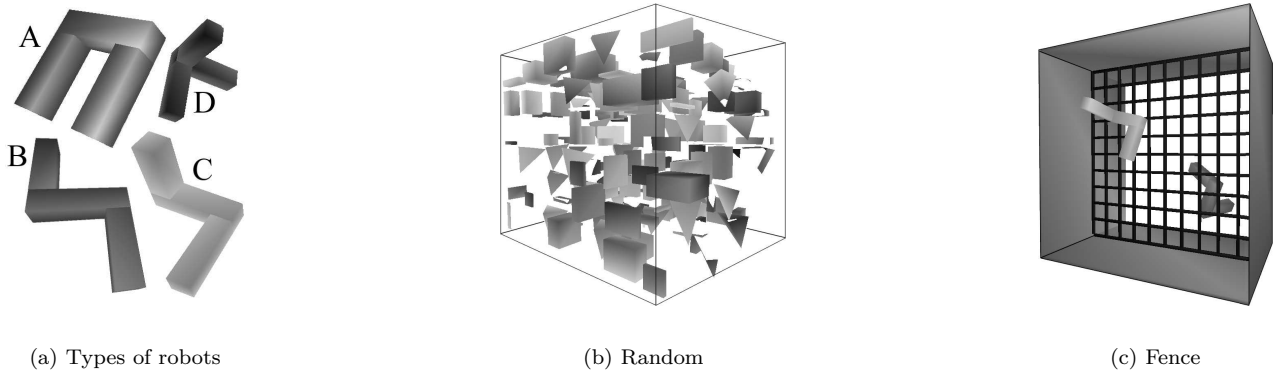
Since the configurations in the RRT are added incrementally and in a spatially local way, an invariant which requires that the tree is fairly well balanced is maintained. The ratio of the point sets on the left and right side cannot be too small or too large. When this occurs at some subtree, the tree is collapsed and rebuilt as a balanced tree. In our experiments, this is necessary but happens sufficiently infrequently that it amortizes well. The trick of not splitting leaves with small point sets greatly reduces the amount of time spent rebalancing trees.

**Queries** Query handling is done by connecting the two query configurations to the roadmap and proceeding by graph search. We find the  $k$ -nearest milestones for each configuration and alternately try to connect them using the BI-RRT algorithm. As soon as both query configurations lie in the same connected component, we cease computation and return the path. We applied some simple path smoothing to the resulting path to improve the quality of the output.

### 3 Methodology and Results

The PRM of RRTs has several parameters which we will now describe:  $N$ , the number of robots,  $n$ , the number of milestones used in the PRM layer,  $m$ , the number of RRT iterations per milestone,  $k$ , the number of iterations that BI-RRT is run for,  $c_{NN}$ , the number of nearest neighbor milestones to use for PRM,  $c_r$ , the number random neighbor milestones to use for PRM, and  $c_{cp}$ , the number of close pairs to check before running BI-RRT.

In our experiments, we measure time and precision. Time is measured for the preprocessing phase (pp time) and for average time to respond to a query



*Figure 2: Robots and scenes for the experiments*

(aq time). We also record the number of SWIFT++ collision detection calls for preprocessing (pp coll.) and per query (aq coll.). Precision is measured by taking some number of non-trivial random queries and counting the fraction solved positively.

The experiments were carried on dual Athlon 1900MPs with 1 gigabyte of RAM. Computation is monolithic, sequential and fits inside the core so there is no disk access.

We tested on a variety of benchmarks (Figure 3). The robots we used are illustrated in Figure 2(a). Problem “fence1” uses one robot of type C and problems “fence2” and “narrow2” use two copies of C. Problem “random3” uses one A, one C and one D. Problem “narrow4” uses one B, two C and one D and we have added one A and one C for “narrow6” and “empty6”. Finally, problems “narrow8” and “empty8” also include one B and one D. The scenes are also shown. In Figure 1, the scene for the narrow passage is shown. In Figure 2(b), the scene filled with random polyhedra is shown. Finally, in Figure 2(c), the scene with the fence is shown.

In Figure 3, we compare three different motion planners. A standard PRM, a BI-RRT, and the PRM of RRTs. Each planner uses the same code base and corresponds to different parameter settings. A PRM is obtained by setting  $m = 0$ ,  $k = 0$  and  $c_{cp} = 1$ . A BI-RRT is obtained by setting  $n = 0$  and  $c_{cp} = 0$ . In setting the remaining parameters, we tried a variety of settings and chose good tradeoffs for each with an emphasis on precision.

## 4 Discussion

In Figure 3, we summarize our results for the various benchmarks. The most important difference that we note is significant increase in reliability over

PRM and over BI-RRT. In our implementation, PRM of RRTs generally outperforms PRM for multiple query problems. Typically, for the cost of two or three BI-RRT queries, we can preprocess the space with PRM of RRTs to obtain a structure which answers queries more robustly and more quickly than BI-RRT. The differences between the methods were more pronounced in the examples with more complex scenes and with more robots. We use fairly standard implementations of RRT and PRM. We think it is likely that improvements to either subroutine would be an improvement for our planner.

The method we present has common attributes with other refinement and non-uniform sampling techniques used in PRM planning. RRTs have a tendency to grow towards obstacles. This gives a tendency to throw more configurations near obstacles which resembles the technique of OBPRM [1]. The rejection sampling and local improvement of RRT is similar to the enhancement phase of early PRM planners [14, 15].

Both RRT and PRM are well-known to be extremely sensitive to the interplay between the metric and the incremental planner [18, 3]. We also made this observation in our implementation. In environments with thin features, in particular the fence environment, BI-RRT tended to produce many configurations that were stuck near obstacles. BI-RRT also makes use of locality and is capable of answering easier queries while avoiding difficult and irrelevant parts of the space. In environments with a single narrow feature, the BI-RRT is forced to do a similar amount of work to the PRM or PRM of RRTs preprocessing phases to answer a single query. This phenomena also accounts for the better performance of BI-RRT on the random example compared to other examples where resolving a query can be often be done without con-

PRM	pp time(s)	pp coll.	aq time(ms)	aq coll.	precision
empty 6	3.75	45202	23.03	217	0.952
empty 8	507.09	1179373	232.19	484.8	0.882
narrow 2	452.8	1030952	307.67	137	0.991
narrow 4	467.79	830600	217.1	148.91	1
narrow 6	1426.32	2040998	269.58	1105	0.242
narrow 8	10544.89	60554186	375.81	710.4	0
random 3	2657.21	3945703	452.63	1130.31	0.544
fence 1	200.83	1671408	503.46	9559	0.597
fence 2	1147	2654280	356.52	1836	0.12125

BI-RRT	pp time(s)	pp coll.	aq time(ms)	aq coll.	precision
empty 6			319.92	9607	0.978
empty 8			4207	75917	0.995
narrow 2			154297	1954355	0.418
narrow 4	<i>n/a</i>	<i>n/a</i>	23250.72	695309	0.357
narrow 6			208022	316989	0.54
narrow 8			1073393	10900158	0.25
random 3			9493.96	94367	0.804
fence 1			4484.52	213279	0.0133
fence 2			254954	344255	0.0

PRM of RRT	pp time(s)	pp coll.	aq time(ms)	aq coll.	precision
empty 6	0.5489	15016	116.1	3200	0.985
empty 8	5.984	103528	1296.75	22261.58	0.97
narrow 2	96.17	8062836	7.426	393.27	1
narrow 4	84.42	3889115	46	2128.45	0.999
narrow 6	421.43	11947585	941.696	26003	0.941
narrow 8	7302.14	10822951	10327	16362	0.68
random 3	25.58	253330	7186	70816	0.961
fence 1	186.46	6713599	56.42	2666.38	1
fence 2	220.63	8789764	143.92	5828	0.875

**Figure 3: Comparison of Planning Algorithms**

sidering the whole space. Finally, we believe that the efficiency of the PRM of RRTs derives in part from offering the BI-RRT calls easier queries as they come from the nearest neighbor clustering.

The algorithms we use are designed to have several opportunities for early exit as a speed enhancement. In collision detection, bisection checking on a path allows for early exit for paths with many collisions. In the PRM layer, only checking edges between different components and halting when all components were determined produced better results. Finally, the BI-RRT algorithm can make an early exit by quickly checking  $k$  close edges before beginning or once a path between the roots has been found. Together these early exit opportunities allow for good overall time improvements without a loss in reliability. Since PRM of RRTs uses all of these opportunities

for early exit, the time improvements are most significant.

In some of experiments, clock time for PRM of RRTs was far superior to PRM even though fewer collision checks occurred in the second method. This occurs for several reasons. Using bisection for collision detection quickly rejects bad edges, however a large number of edge checks are initiated. The incremental planner will continue checking an edge once a failure occurs. PRM of RRTs checks fewer edges but works harder for each edge. This is reflected in the running time. Also, the nearest neighbor queries lead to super-linear growth in the running time. On more difficult examples, PRM needs many milestones to succeed and the  $k$ -D tree has many points in it. As the number of points in the tree grows, this cost begins to dominate the running time since it is the

only super-linear cost in the implementation. The hierarchical representation of PRM of RRT yields much smaller trees and this problem does not manifest as seriously.

The results we present were for good parameter selection for each method. In a new example, selecting the correct parameters can be difficult, particularly for PRM of RRTs since the tradeoff between the number of milestones and number of BI-RRT iterations is more sensitive. In general, generating no more than several hundred initial milestones and using dense PRM seemed to be the best setting. Once this amount is fixed, varying the number of BI-RRT iterations generates a tradeoff between precision and time. The incremental planner, metrics and C-space representation can be varied to optimize performance as with other motion planners using similar frameworks. We showed how to provide parameters that yield a smooth spectrum between the PRM and BI-RRT approaches. The planner we obtained was effective for high degree of freedom problems obtained by putting multiple non-convex rigid robots in various scenes. In many cases the advantages of this technique were striking. Although further experimentation is needed to better understand parameter sensitivity, we believe the framework we describe is useful as we attempt to solve increasingly difficult planning problems.

**Acknowledgements** Work on this paper by K. Bekris, B. Chen, A. Ladd, E. Plaku and L. Kavraki has been partially supported by NSF 9702288, NSF 020567 a Whitaker Grant, and a Sloan Fellowship to L. Kavraki. A. Ladd is also partially supported by an FCAR grant.

## References

- [1] N. Amato, B. Bayazit, L. Dale, C. Jones, and D. Vallejo. Obprm: An obstacle-based prm for 3d workspaces. In P. Agarwal, L. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 156–168. AK Peters, 1998.
- [2] N. Amato, K. Dill, and G. Song. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. pages 2–11, April 2002.
- [3] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Trans. Robot. & Autom.*, pages 442–447, 2000.
- [4] M. Apaydin, D. Brutlag, C. Guestrin, D. Hsu, and J. Latombe. Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion. In *International Conference on Computational Molecular Biology (RECOMB)*, April 2002.
- [5] J. Barraquand and J. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10:628–649, 1991.
- [6] M. de Berg, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 1997.
- [7] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *J. of the ACM*, 40:1048–1066, 1993.
- [8] S. Ehmann and M. Lin. Accurate and fast proximity queries between polyhedra using surface decomposition. *Computer Graphics Forum (Proc. of Eurographics)*, 2001.
- [9] R. Geraerts and M. Overmars. A comparative study of probabilistic roadmap planners. In *Proc. WAFR 2002*, 2003.
- [10] D. Hsu, R. Kindel, J. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Int. J. Robot. Res.*, 2001.
- [11] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE Int. Conf. Robot. & Autom.*, pages 2719–2726, 1997.
- [12] Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Trans. Robot. & Autom.*, 8(1):23–32, Feb. 1992.
- [13] P. Isto. Constructing probabilistic roadmaps with powerful local planning and path optimization. In *IEEE/RSJ Int. Workshop on Intelligent Robots & Systems (IROS)*, 2002.
- [14] L. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford University, 1995.
- [15] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. & Autom.*, 12(4):566–580, June 1996.
- [16] A. Ladd and L. Kavraki. A measure theoretic analysis of prm. In *IEEE International Conference on Robotics and Automation*, May 2002.
- [17] A. Ladd and L. Kavraki. Motion planning for knot untying. In *International Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2003.
- [18] S. LaValle and J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *International Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2000.
- [19] S. LaValle and J. Kuffner. Randomized kinodynamic planning. *Int. J. Robot. Res.*, 5:348–400, May 2001.
- [20] S. LaValle and J. Kuffner. Rapidly exploring random trees: Progress and prospects. In B. Donald, K. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A.K. Peters, 2001.
- [21] T.-Y. Lie and Y.-C. Shie. An incremental approach to motion planning with roadmap management. In *IEEE Int. Conf. Robot. & Autom.*, 2002.