2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Single-Query Motion Planning with Utility-Guided Random Trees

Brendan Burns Oliver Brock
Department of Computer Science
University of Massachusetts Amherst

Abstract— **Randomly expanding trees are very effective in exploring high-dimensional spaces. Consequently, they are a powerful algorithmic approach to sampling-based single-query motion planning. As the dimensionality of the configuration space increases, however, the performance of tree-based planners that use uniform expansion degrades. To address this challenge, we present a utility-guided algorithm for the online adaptation of the random tree expansion strategy. This algorithm guides expansion towards regions of maximum utility based on local characteristics of state space. To guide exploration, the algorithm adjust the parameters that control random tree expansion in response to state space information obtained during the planning process. We present experimental results to demonstrate that the resulting single-query planner is computationally more efficient and more robust than previous planners in challenging artificial and real-world environments.**

I. Introduction

Random tree expansion is a successful algorithmic approach to sampling-based single-query motion planning. A single-query planning problem is defined by two points in the state (or configuration) space of a robot. A single-query planner attempts to connect these states by a sequence of valid transitions through state space. The success of random tree expansion planners in the single-query planning domain can be explained by two advantageous characteristics of random tree expansion: *1)* random trees rapidly explore state space, leading to an effective search for a solution to the planning problem, and *2)* the exploration of state space can be performed based on control inputs, making the algorithm applicable to systems that are subject to differential constraints.

A large number of random tree expansion planners have been proposed [1], [2], [3], [4], [5], [6] and applied in challenging domains, ranging from molecular biology [7] to virtual prototyping [8]. As in all of motion planning, the performance of random tree expansion planners degrades as the dimensionality of the state space increases. Consequently, numerous recent methods improve the performance of tree-based planning by adapting expansion based on local features of the state space [9], [10], [11]. These planners leverage the structure of configuration space to render the exploration of state space more efficient. Given that the provable difficulty of motion planning cannot be overcome in the general setting, leveraging of information contained in a particular instance of the planning problem is a promising approach to improving the performance of tree-based planners in high-dimensional state space.

In this paper we introduce a utility-guided approach to random tree exploration. The notion of utility has successfully been applied in multi-query sampling-based motion planning [12]. Here it enables effective online adaptation of the exploration behavior of random tree planners. As our experimental results indicate, a single-query planning method based on this approach leads to significant performance improvements relative to previous tree-based planners. These improvements become more pronounced as the dimensionality of the state space increases.

To achieve these performance improvements, we first present a compressive algorithmic framework for tree-based single-query planning. The framework is sufficiently general to express existing tree-based planners as instantiations. It also reveals the fundamental parameters that govern the expansion behavior of random trees. We propose an algorithmic approach to controlling expansion behavior by adjusting these expansion parameters throughout the planning process. The approach is based on notion of expected utility from Bernoullian utility theory [13]. Given the state space information obtained in previous expansion steps of the planner, our approach maximizes the expected utility of the next expansion step. The resulting planning framework directs state space exploration so that—given the available information—maximum expected progress towards a solution is made.

II. Random Tree Exploration and Related Work

We begin by presenting an algorithmic framework for random tree exploration algorithms. This framework exposes a set of modular components that are part of all tree-based motion planners presented in the literature. The framework will structure our discussion of related work in this section, but it will also serve as the scaffolding for implementing our utility-guided random tree planners in Section III.

Tree-based planners explore space by incrementally expanding a tree of state transitions, starting from the initial and the goal state of the robot. During the expansion phase, an existing node of a tree is selected for expansion. The expansion is performed by adding a new node to the tree. The new node represents a single exploration step. To determine the node, the algorithm chooses a direction and a distance for the exploration. After each successful expansion, the planner attempts to connect the two random trees. If this connection succeeds, a path is found. Otherwise the expansion continues. This basic algorithmic framework, consisting of node selection, expansion direction selection, expansion length selection, and the connection attempt, is summarized in Figure 1.

```
RandomTreePlanner($q_{\text{start}}, q_{\text{goal}}$)
1:    $A = \text{Tree}(q_{\text{start}})$
2:    $B = \text{Tree}(q_{\text{goal}})$
3:    while ( path not found )
4:        $q$ = SelectNode($A$)
5:        $\vec{d}$ = SelectDirection($A$, $q$)
6:        $\delta$ = SelectDistance($A, q, \vec{d}$)
7:        if (Extend($A, q, \vec{d}, \delta$))
8:          if (Connect($A, B$))
9:            return path
10:       Swap($A, B$)
```

Fig. 1. The generalized random tree expansion algorithm

This basic tree-based exploration algorithm identifies four algorithmic components that determine the expansion behavior of the trees generated by the planner: node selection, direction selection, distance selection, and connection. All existing planners can be described by choosing particular implementations for these components. These implementation choices are summarized in Table I summarizes for several existing tree-based planners.

One of the first algorithms for random tree planning is the Z^3 [1]. The algorithm grows a tree from start to goal through a series of randomly selected sub-goals. Paths connecting nodes in the tree are computed using a local planner that attempts local obstacle avoidance. This planner only constructs a single tree.

The most widely used tree-based planners are based on rapidly-exploring random trees (RRTs). The original RRT-Connect [3] algorithm uses a Voronoi bias to select expansion node and direction. More recent research has refined the Voronoi bias using the (adaptive) dynamic domain [9], [10] which limits the expansion of nodes near obstacles.

Much additional related work is summarized in Table I. This table also illustrates that the largest amount of research has explored the node selection for expansion and, to a lesser extent, the direction of this expansion. Relatively little research has examined the role of the exploration length or the algorithm for connecting growth. The approach proposed in the next section provides a general approach to the adaptation of all of the parameters of tree-based exploration.

III. Utility-Guided Random Trees

We now introduce utility-guided random tree exploration and present an implementation of a utility-guided single-query planner.

A. *Utility-guided exploration*

Let the set E be the set of possible expansions e_i for a given random tree. We would like to choose the expansion with maximal expected utility, i.e., the expansion step that is expected to provide maximum progress towards a solution to the planning problem. If we estimate the utility Utility(e_i) as well as the probability $P(e_i)$ that e_i succeeds and leads to an expansion of the tree, we can compute the expected utility [13] of e_i as

$$\text{ExpectedUtility}(e_i) = P(e_i)\ \text{Utility}(e_i).$$

By always choosing the expansion step with the largest expected utility, we can maximize progress of the planner. Of course, we first have to provide estimates for $P(e_i)$ and Utility(e_i). The effectiveness of a utility-guided planner will critically depend on the accuracy of these estimates. In the context of multi-query planning, our previous work has shown that even coarse estimates provide significant performance improvements [19].

In the following we present an implementation of a single-query planner based on utility-guided tree exploration. The next four sections describe estimators of utility for the four main algorithmic components of random tree exploration: node selection, exploration direction, exploration distance, and the connection attempt (see Figure 1). These four utility estimators are invoked sequentially in the respective functions during the execution of the general algorithm. We assume that the utility of each step in algorithm is independent. Section III-F describes how to estimate $P(e_i)$ using a non-parametric model of configuration space. Both the utility estimators and the configuration space model can be easily replaced with alternative implementations, which will be the subject of future investigations.

B. *Utility of node for expansion*

The first step in random tree expansion selects the node to be expanded. The utility of selecting a particular node is tied to the possibilities for meaningful exploration of configuration space connectivity still possible from that node. Of course, without a precise understanding of the state of the configuration space surrounding a node this is impossible to know. We estimate this quantity to be inversely proportional to the number of exploration attempts originating from the node. This estimator predicts high utility for nodes on the fringe of the tree and a uniform, lower utility for interior nodes that are far removed from the leaves. By choosing other estimators for this utility, it is possible to change the balance between exploration and refinement during tree growth. An additional benefit of this node selection strategy is that it can be implemented in constant time, avoiding the expensive nearest-neighbor queries used by other approaches such as the Voronoi bias.

C. *Utility of expansion direction*

Once a node has been selected, a direction for expansion must be chosen. Because the goal of this expansion is exploration, the most useful directions for exploration are those most different from previous explorations originating from the node. Let D be the set unit vectors $\vec{d_i}$ representing

Algorithm	Node Selection	Expl. Direction	Expl. Length	Connection	# Trees
Z^3 [1]	Uniform Random	Local Planner	Local Planner	N/A	1
Ariadne's Clew	Fixed	Local Planner	Local Planner	Local Planner	1
RRT-Connect [14]	Voronoi Bias (V.B.)	Voronoi Bias	Constant	Nearest Node	2
DD-RRT [9]	DD V.B.	DD-V.B.	Constant	N/A	1
ADD-RRT [10]	ADD V.B.	ADD-V.B.	Constant	Nearest Node	2
OB-RRT	Voronoi Bias	Hybrid	Hybrid	Nearest Node	2
Blossom-RRT [11]	Voronoi Bias	All	All	N/A	1
SBL [6]	C-Space Density	Uniform Random	Shrinking Neighborhood	Bridge Node	2
Exp. Spaces [15]	Tree density	Tree density	$<$ Constant	$<$ Constant	2
Guided Exp. Spaces [16]	Heuristic	Heuristic	$<$ Constant	$<$ Constant	2
Adapt. Single-Query [5]	none	none	none	Heuristic	2
RRFT [17]	Dyn. or Hist. Based	Adapt. Goal Biased	$<$ Constant	N/A	1

TABLE I
SUMMARY OF TREE-BASED PLANNERS PRESENTED IN THE LITERATURE

previous expansion directions for a node q. The utility of of some new expansion direction $\vec{d}$ for this node is given by

$$\text{Utility}(\vec{d}) = \sum_{\vec{d}_i \in D} -\text{success}(\vec{d}_i)\ (\vec{d} \cdot \vec{d}_i),$$

where $\vec{d} \cdot \vec{d}_i$ is the dot product between the two unit vectors, indicating the cosine of the angle between them. The value success$(\vec{d}_i)$ is used to bias selection towards directions that are likely to be unobstructed. In our implementation this function returns a constant c for directions $\vec{d}$ that resulted in a successful expansion, and $c/2$ otherwise. Based on this function, the planner uses past experience to avoid subsequent expansions that are likely to be obstructed.

D. Utility of exploration distance

The utility estimators for node selection and direction selection do not rely on estimates of probability. The probability in both cases is always one, since every node selection and every direction selection will always be successful. The remaining two components of the algorithm, exploration distance and connection attempt, traverse the state space and therefore have to consider the probability of this traversal being successful. A traversal that is unlikely to be successful has low expected utility.

The remaining two components of the utility-guided algorithm, namely the exploration step during tree expansion and the attempt to connect the two trees, are performed in a very similar fashion. However, there is one important distinction: during exploration the exploration direction is chosen as described in Section III-C, whereas the expansion direction during the connection attempt is given by the state of the other tree. We begin by describing the process of determining the exploration distance.

At this point, the algorithm has selected a node q and a direction $\vec{d}$ for the expansion. We now need to determine a distance δ for this expansion. Assuming that δ has been determined, the planner then perform collision checks in increments of ϵ along the expansion direction to validate that the expansion is valid, until the distance δ. The planner could determine δ based on the information available at this point of the planning process. However, it would be advantageous to determine it incrementally by exploring the state space along $\vec{d}$. This will allows us to gather additional information about the state space in the expansion direction, leading to improved estimates of utility.

To determine δ, we proceed in increments of $\alpha >> \epsilon$ along $\vec{d}$. At each α increment, we evaluate the expected utility of that point a'. If this expected utility exceeds a threshold u_{min}, the algorithm decides to expand to that point and δ is temporarily set to the distance between q and q'. The planner now validates the connection between q and q' in ϵ increments. This validation obtains more information about local state space features that help to estimate the utility at the next α increment along $\vec{d}$. This process continues until the utility of q' is below the threshold or the connection between two states is invalid.

The planner estimates the expected utility of a particular point q' based on the probability of it being collision free and its utility. The former can be obtained from the state space model described in Section III-F. To determine the utility of a point n', we consider its distance δ to q, the node being expanded. If this distance is larger than a threshold δ_{max}, the utility of q' is zero, otherwise it is identical to the distance δ itself. The introduction of this cut-off is motivated in Figure III-D. The exploration step of the algorithm is intended to move out of the shadow of state space obstructions so that the connection step (described next) has an increased chance of successfully connecting the two trees. If the exploration length is too large, the likelihood of connecting the two trees is not improved and the cost of validating the transition becomes very large.

Other researchers have also identified excessively long exploration steps as problematic for several single-query planners [14], [20], thus providing support for our choice of utility function.

E. Utility of connection attempt

Attempting to connect the two trees proceeds in a similar fashion as the determination of the expansion length. Now, the state q' obtained by the expansion step itself is used for

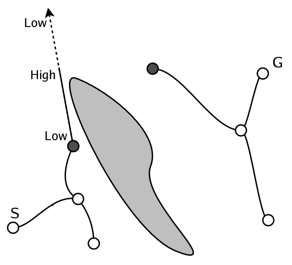

Fig. 2. An illustration of utility for expansion distance. Expansion beyond the shadow of the obstacle does not increase the utility of the expansion.

expansion. The expansion direction $\vec{d}$ is given by the direction towards the closest node of the other tree. The definition of utility does not include the cut-off, since the connection step attempts to connect the trees in a greedy fashion. The expected utility of an increment in the connection attempt can only become smaller than the minimum threshold u_{min}, if the probability of n' being collision-free becomes very small. Effectively, the connection step proceeds in a greedy fashion with an α look-ahead based on the state space model.

F. Modeling Configuration Space

Above, we have relied on an estimate of the probability that a configuration or edge in state space is valid or collision free. This is accomplished with the use of a non-parametric model [19]. This model stores the outcome of all state evaluations in a bag. Based on this information it is able to make predictions about the state evaluation of previously unseen states. To make such a prediction for an unseen state q, we determine the state's k nearest neighbors stored in the model. Based on the validity of q's neighbors and their distances to q, we can compute a prediction for the validity of q. It is noteworthy that this predictive, non-parametric model stores positive and negative state evaluations and is thus able to leverage all information obtained from previous state evaluations. Because a Euclidean distance metric has known problems in high-dimensional spaces, we use an alternate distance metric proposed by Leven et al. [21] for measuring the distance between configurations.

IV. Experiments

The effectiveness of random tree planning depends on the planner's ability to rapidly explore relevant state space. This ability in turn depends on how well the planner can adapt the exploration to local features of the state space. To demonstrate the effectiveness of utility-guided random tree planning in that regards, we compare its performance with adaptive dynamic-domain RRT [10], a state-of-the-art random tree planner (AD-RRT in Figure 4). The adaptive-domain RRT represents a recent, advanced RRT-based planner and is well-suited to serve as a comparison. For AD-RRT we used an adaptive factor of 0.95 and a radius of 20 times ϵ, as suggested in the original paper. We do not report here on experiments with the regular RRT algorithm, which in many cases was unable to solve the planning problems within the allotted amount of time.

To elucidate the role of utility-guided adaptation of exploration, we employ two different utility-guided algorithms: the first is a hybrid algorithm that uses the traditional Voronoi bias for selecting nodes and exploration direction, but picks expansion distance and connection attempts based on utility (Vor/Util-RRT). The second algorithm is the fully utility-guided random tree algorithm described in the previous section (Util-RRT).

The performance of these three algorithms is compared in two different types of planning problems. The "bug trap" (Figure 3), which has recently been identified as a challenging benchmark for RRT methods [10], [9], serves as the first planning problem. Whereas previously these bug traps were only considered in two dimensions, we also consider higher-dimensional bug traps. In a second set of experiments, we compare planning performance for reaching tasks on a 14 degree-of-freedom humanoid platform.

A. Bug trap experiments

A bug trap of arbitrary dimension can be specified by a hyper-sphere shell centered around the origin. The shell is pierced by an un-capped hyper-cylinder of the same thickness oriented along the x-axis and extending from the origin to the edge of the hyper-sphere. If a configuration is within the width of the shell or the hull of cylinder it is considered obstructed; otherwise it is free.

Previous work has identified a strong correlation between the relative size of the bug trap and planner performance [10], [9]. Bug traps of various relative sizes are shown in Figure 3. Larger bug traps are much easier to solve than smaller bug traps because a certain degree of refinement (as opposed to expansion) has to be achieved to find the narrow passage.

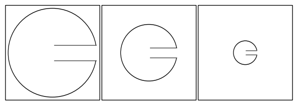

Fig. 3. Bug traps of different sizes (large, medium, small), relative to the considered configuration space

To evaluate the effectiveness of utility-guided random trees in adapting exploration to local features of the state space, we tested all three planners on bug traps of varying dimensionality (ranging from two to five) and of varying relative sizes (large, small, medium). In each world, each of the planners was asked to compute a path between the same random point inside the bug trap to the same random point outside the bug trap. The length of time to compute this path was recorded. The average results for 50 path queries with 50 different random point pairs are shown in Figure 4. For each experiment we kept track of the average run time (top row of graphs in Figure 4) of the planners as well as whether they successfully solved the planning problem (bottom row of graphs in Figure 4). If a planner did not find a solution to the planning problem after five minutes, we considered the

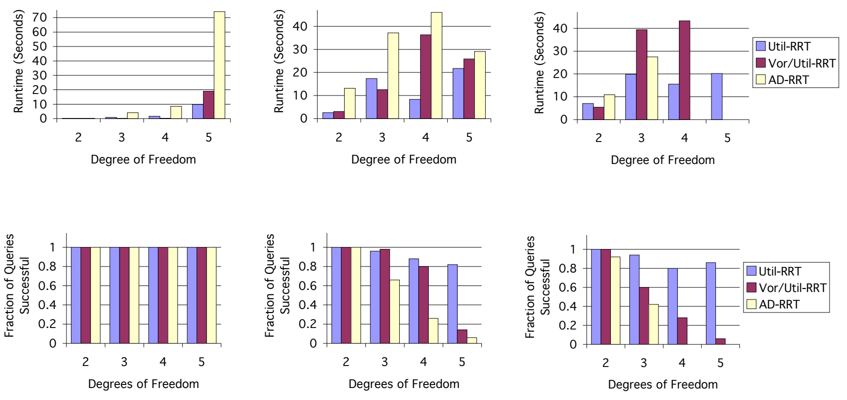

Fig. 4. Average run times (top row) and success rates (bottom row) for different algorithms on large, medium, and small bug traps (relative to the configuration space size)

DOF	AD-RRT	Hybrid Vor./Util. RRT	Full Utility RRT
2	0.14 (0.21)	0.14 (0.39)	0.17 (0.41)
3	4.06 (9.26)	0.23 (0.23)	0.83 (1.52)
4	8.56 (14.20)	0.37 (0.38)	1.63 (4.12)
5	74.17 (163.92)	19.01 (31.73)	9.79 (21.24)

TABLE II

AVERAGE RUN TIME (STANDARD DEVIATION) FOR LARGE BUG TRAPS

experiment failed. The numerical results for running times in the case of the large bug trap are shown in Table II.

The experimental results show that utility-guided random tree planning significantly outperforms the adaptive-domain RRT planner. The performance improvements become more pronounced as the dimensionality of the bug trap increases. The performance improvements also increase as the planning problem becomes more difficult when the relative size of the bug trap is reduced. For the medium and small bug traps, the fraction of successful planning attempts is also improved, in particular as the dimensionality of the bug trap increases. In all experiments, the utility-guided random tree planner never falls below 80% success rate, while the other two algorithms drop below 10% success rate. The fully utility-guided planner consistently outperforms the hybrid planner, further validating the importance of utility-guided expansion.

These results indicate that utility-guided random trees are more effective than existing planners at adapting the exploration of state space based on the partial information obtained about local state space features. As a result, the utility-guided random tree planner is both more efficient and more reliable. These attributes are necessary for effective real-world motion planning.

There are some cases in which the hybrid planner slightly outperforms the fully utility-guided planner (two-dimensional large bug trap, three-dimensional medium bug trap, and two-dimensional small bug trap). This can be explained by the simplicity of the environment considered here. The cost of performing a collision check for a bug trap is very small. We suspect therefore that in these cases the additional computational cost of determining the utility exceeds the gains obtained from more effective exploration. Our real-world experiments in the next section confirm this hypothesis.

B. Real World Experiments

The experiments in bug trap worlds demonstrate that utility-guided random tree planning can improve performance in carefully constructed, challenging configuration spaces. From a practical perspective it is important that the algorithm also improve the performance of planning for real-world robots. Thus, we examined planner performance in the context of two related real-world tasks for a 14-DOF humanoid torso (Figure 5). Both tasks examine the assembly of pipes in an obstructed environment, modeling an assembly and service task on the exterior of the International Space Station. The motion planning problems consist of a start position with the arms extended on the outside of a box in which the assembly will take place. The goal position is inside the box with the pipes oriented and ready for a lower-level force-controller to orchestrate the final assembly. To experiment with our motion planner on problems of varying difficulty, we performed experiments with and without a lid on the box. Images of the start and goal for both of these experiments are shown in Figure 5. For each of these experiments, we performed 50 different requests for the same path query to planners seeded with different random seeds. The average run time and standard deviation for each planner and each task are given in Figure 6.

These results show that utility-guided planning is well suited to real-world motion planning. The hybrid planner is nearly twice as fast as existing state-of-the-art techniques. The entirely utility-guided planner is between two and four times as fast. The performance improvement of complete utility-guided planning increases as the complexity of the problem increases. This demonstrates that in addition to improving run time performance, utility-guided planning scales

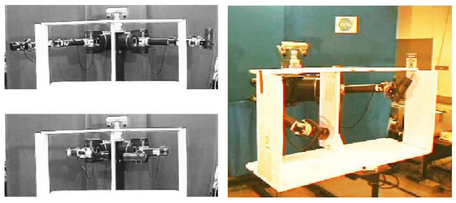

Fig. 5. The humanoid robot Dexter in its start and goal configurations (left) and enacting a successful motion plan (right)

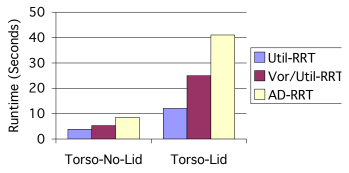

Fig. 6. Average run time of various planning algorithms for the 14-DOF humanoid torso in two different environments

better with respect to the complexity of the underlying problem. Also of significance are the run times themselves. In a moderately complex world (the box without the lid) utility-guided planning has an average run time that is adequate for planning in slowly changing dynamic environments.

V. Conclusion

The computational efficiency of random tree single-query planners critically depends on their ability to adapt state space exploration based on local features of state space. If the information obtained during planning can be effectively leveraged to guide exploration towards regions relevant to the path query, the volume of state space that must be explored is reduced, resulting in more efficient planning.

We have proposed a novel utility-guided single-query planner. This planner guides the ongoing tree-based exploration of state space using information about state space obtained from previous tree expansions. Our planner incrementally learns how to adjust the parameters of tree-based exploration based on the structure of state space that is revealed during the planning process. To achieve this, the planner identifies expansion steps with maximal expected utility given its current knowledge of the state space. Thus, the planner uses available information to maximize expected progress towards a successful path.

Experimental results in both challenging artificial and real-world environments indicate that this new utility-guided planner is faster and more reliable than existing state-of-the-art random tree planners. The performance improvements over tree-based planners presented in the literature increase with the complexity of the planning problem and the dimensionality of the configuration space, indicating that the benefit of utility-guided exploration increases with the difficulty of the planning problem.

Acknowledgments

This work is supported in part by the National Science Foundation (NSF) under grants CNS-0454074, IIS-0545934, and MIT/NASA cooperative agreement NNJ05HB61A.

References

[1] B. Baginski, "Local motion planning for manipulators based on shrinking and growing geometry models," in *Proceedings of the International Conference on Robotics and Automation*, vol. 4, 1996, pp. 3303–3308.

[2] D. Hsu, L. E. Kavraki, J.-C. Latombe, and R. Motwani, "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. Kluwer Acedemic Publishers, 1999, pp. 159–182.

[3] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *Proceedings of the International Conference on Robotics and Automation*, Detroit, USA, 1999.

[4] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings of the International Conference on Robotics and Automation*, vol. 2, San Francisco, USA, 2000, pp. 995–1001.

[5] D. Vallejo, I. Remmler, and N. M. Amato, "An adaptive framework for single shot motion planning: A self-tuning system for rigid and articulated robots," in *Proceedings of the International Conference on Robotics and Automation*, Seoul, Korea, 2001, pp. 21–26.

[6] G. Sánchez and J.-C. Latombe, "On delaying collision checking in prm planning: Application to multi-robot coordination," *International Journal of Robotics Research*, vol. 21, no. 1, pp. 5–26, 2002.

[7] J. Cortés, T. Siméon, V. Ruiz de Angulo, D. Guieysse, M. Remaud-Siméon, and V. Tran, "A path planning approach for computing large-amplitude motions of flexible molecules," *Bioinformatics*, vol. 21, no. Suppl. 1, pp. i116–i125, June 2005, proceedings of the International Conference on Intelligent Systems for Molecular Biology (ISMB), Detroit, USA.

[8] E. Ferré and J.-P. Laumond, "An iterative diffusion algorithm for part disassembly," in *Proceedings of the International Conference on Robotics and Automation*, New Orleans, USA, April 2004, pp. 3149–3154.

[9] L. Jaillet, A. Yershova, S. LaValle, and T. Simeon, "Adaptive tuning of the sampling domain for dynamic-domain RRTs," in *Proceedings of the International Conference on Intelligent Robots and Systems*, 2005.

[10] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," in *Proceedings of the International Conference on Robotics and Automation*, 2005.

[11] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato, "An obstacle-based rapidly-exploring random tree," in *Proceedings of the International Conference on Robotics and Automation*, Orlando, FL, May 2006.

[12] B. Burns and O. Brock, "Toward optimal configuration space sampling," in *Proceedings of the Robotics: Science and Systems Conference*, Cambridge, Massachusetts, 2005.

[13] N. E. Jensen, "Introduction to bernoullian utility theory," *Swedish Journal of Economics*, pp. 163–183, 1967.

[14] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, 2000, pp. 293–308.

[15] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry and Applications*, vol. 9, no. 4, pp. 495–512, 1999.

[16] J. Phillips, N. Bedrossian, and L. Kavraki, "Guided expansive spaces trees: A search strategy for motion- and cost-constrained state spaces," in *Proceedings of the International Conference on Robotics and Automation*, New Orleans, LA, April 2004.

[17] J. Kim, J. M. Esposito, and V. Kumar, "An rrt-based algorithm for testing and validating multi-robot controllers," in *Proceedings of the Robotics: Science and Systems Conference*, 2005.

[18] P. Bessière, J.-M. Ahuactzing, E.-G. Talbi, and E. Mazer, "The ariadne's clew algorithm: Global planning with local methods," vol. 2, 1993, pp. 1373–1380.

[19] B. Burns and O. Brock, "Entropy-guided single-query motion planning," in *Proceedings of the International Conference on Robotics and Automation*, 2005.

[20] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proceedings of the International Conference on Robotics and Automation*, vol. 1, San Francisco, USA, 2000, pp. 521–528.

[21] P. Leven and S. Hutchinson, "Toward real-time path planning in changing environments," in *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, 2000.