Adaptive Tuning of the Sampling Domain for Dynamic-Domain RRTs*

Léonard Jaillet† Anna Yershova‡ Steven M. LaValle‡ Thierry Siméon†

†*LAAS-CNRS*
7, Avenue du Colonel Roche
31077 Toulouse Cedex 04,France
{*ljaillet, nic*}*@laas.fr*

‡*Department of Computer Science*
University of Illinois
Urbana, IL 61801 USA
{*yershova, lavalle*}*@uiuc.edu*

***Abstract*— Sampling based planners have become increasingly efficient in solving the problems of classical motion planning and its applications. In particular, techniques based on the Rapidly-exploring Random Trees (RRTs) have generated highly successful single-query planners. Recently, a variant of this planner called dynamic-domain RRT was introduced in [28]. It relies on a new sampling scheme that improves the performance of the RRT approach on many motion planning problems. One of the drawbacks of this method is that it introduces a new parameter that requires careful tuning.**

In this paper we analyze the influence of this parameter and propose a new variant of the dynamic-domain RRT, which iteratively adapts the sampling domain for the Voronoi region of each node during the search process. This allows automatic tuning of the parameter and significantly increases the robustness of the algorithm. The resulting variant of the algorithm has been tested on several path planning problems.

***Index Terms*— Motion Planning, Voronoi Bias, RRTs.**

I. Introduction

Sampling-based approaches in motion planning have solved many difficult problems in recent years. They generally can be divided into two sets of approaches: multiple-query and single-query methods. The philosophy behind the multiple-query methods is that substantial precomputational time may be taken so that multiple queries for the same environment can be answered quickly. The Probabilistic Roadmap (PRM) planners [1], [4], [11], [16], [24] are examples of such method. There are many extensions and adaptations of PRM framework to different instances of motion planning problems, such as solving problems with narrow corridors [9], [12], planning for closed chains [6], [10], [27], multiple robots [26], and nonholonomic robots [23].

Multiple-query methods may take considerable precomputational time, thus different approaches were developed for solving single-query problems [3], [13], [19], [21], [22]. In particular, techniques based on the Rapidly-exploring Random Trees (RRTs) have been highly successful in many applications of motion planning [19], [20], [20]. They have been used to handle complicated geometries [5], [7], manipulation problems and motions of closed articulated chains [6], [14], [15], kinodynamic and nonholonomic planning [8], [17], [18], [20]. Even though RRTs work well on many problems, there are cases when they perform poorly. This happens when expensive operations are executed in each iteration of the RRT planner, while a little progress is done in expansion of the tree.

Recently, a new planner called dynamic-domain RRT has

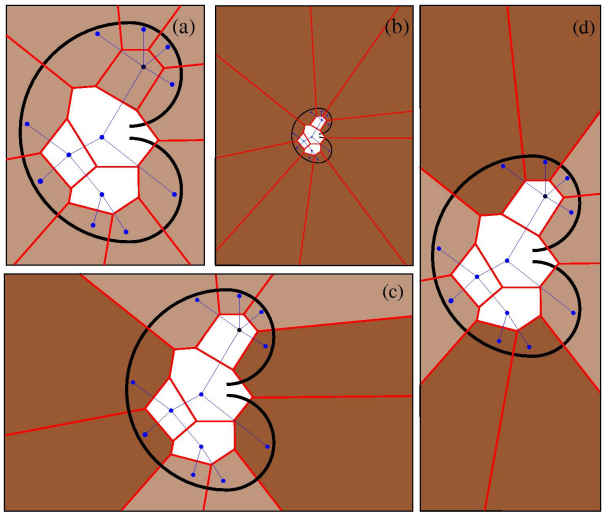


Fig. 1. For the RRT algorithm, the Voronoi region of the frontier nodes is growing together with the size of the configuration space. Therefore, depending on the boundaries of the space, the bias toward unexplored regions can be small (a), strong (b) or biased toward only some of the parts of the space (c and d).

*This work is partially supported by NSF CAREER Award IRI-9875304, NSF ANI-0208891, NSF IIS-0118146, the UIUC-CNRS grant, and the European project MOVIE, ST-20001-39250.

been developed [28], that significantly outperforms other existing RRT-based planners on many motion planning problems. The sampling scheme of the planner takes into account the obstacles of the configuration space into its Voronoi bias. This helps to significantly reduce the number of iterations needed in order to find a solution path. As a result, problems with complicated geometries can be solved by orders of magnitude faster.

Originally, dynamic-domain RRTs were proposed to have an additional parameter, which corresponds to the size of the sampling domain. The performance of the algorithm relies on careful tuning of this parameter for each particular motion planning problem, which makes the method more difficult to use comparing to the original RRTs. In this paper we propose an improved version of the dynamic-domain RRT which automatically tunes the input parameter therefore yielding a more robust performance. We evaluate the proposed method on various classical motion planning problems.

In the next section we review the basic RRT method. In Section III we consider the original dynamic-domain RRTs. We investigate the influence of the radius parameter of the dynamic domain on the exploration properties of the resulting trees in section IV. The method for automatic tuning of the parameter is introduced in Section V and the experimental results for it in Section VI.

II. The RRT framework

A. General approach

RRTs were originally introduced in [19]. Starting at a given initial configuration, RRTs incrementally grow a tree to explore the configuration space and find a path connecting the initial and a goal configurations. The method is based on a simple heuristic: at each iteration a new configuration is sampled at random and the expansion from the nearest node in the tree toward this sample is attempted. If the expansion succeeds, a new node is added to the tree.

Several variations of this planner exist which exploit the same exploration properties. In the basic RRT-Extend algorithm, at each iteration a single expansion step of fixed distance is performed. In a more greedy variant, RRT-Connect [20], the expansion step is iterated while keeping feasibility constraints (e.g. no collision exists). Bidirectional versions of RRTs also exist (bi-RRTs), which alternate execution of the basic algorithm for two trees growing from the initial and the goal configurations, and put some additional bounds on the sizes of each of the trees (bidirectional balanced RRTs).

RRTs exploration is determined by the Voronoi diagram of the nodes in the tree. The probability that a node will be chosen for an expansion is proportional to the volume of its Voronoi region. Therefore, the RRTs tend to rapidly grow in the unexplored regions of the configuration space. Note, that this intrinsic property of the algorithm does not require an explicit computation of the Voronoi diagram.

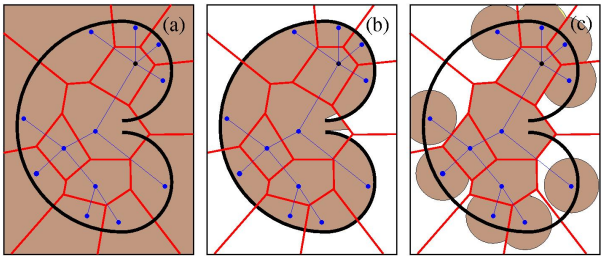


Fig. 2. For a set of points inside a bug trap different sampling domains are shown: (a) regular RRTs sampling domain, (b) visible Voronoi region, (c) dynamic domain.

B. Dependence on the configuration space boundaries

Consider a tree developed by an RRT-based planner. Nodes for which their Voronoi regions grow together with the size of the configuration space (all gray/brown regions in Figure 1) are called *frontier* nodes. In classic RRT planners the sampling domain is defined by the boundaries of the configuration space. Since the behavior of the planner depends on the Voronoi regions of the frontier nodes, it also depends on the particular setting of these boundaries. If these boundaries are chosen unsuitably to a given motion planning problem the efficiency of the planner can be dramatically affected.

Classically, at the beginning of the expansion, when the volume of the configuration space is significantly bigger than the area covered by the tree, frontier nodes provide especially strong bias toward unexplored portions of the configuration space. Often this helps the tree to rapidly grow. However, this may cause a slow-down in the performance when a frontier node is also a *boundary* node, i.e. when it lies in some proximity to the obstacles. Such nodes have a high probability of being chosen for expansion in the direction of the obstacles, but most of the time the expansion fails.

This problem is illustrated by the example in Figure 1. The task is to move the robot outside of a bug trap obstacle. Because of the narrow passage, this problem can be challenging for any motion planner. For classic RRT algorithm the difficulty also comes from the fact that frontier nodes quickly become boundary nodes. Thus, the performance of the planner highly depends on how the boundaries of the environment have been set. In Figure 1 (a), the boundaries of the environment are close to the coverage of the tree. Therefore, it gives quite a good chance to refine the existing tree inside the explored region and thus to solve the problem. In (b) the sampling domain is much bigger than the size of the tree. In this case it will lead to many useless attempts of expansion of the frontier nodes toward the obstacles. Consecutively, it

will decrease the performances of the planner.

Examples (c) and (d) illustrate another situation leading to an arbitrary bias toward only some of the directions of the configuration space. To overcome this drawback, the DD-RRT algorithm proposes a way to control the Voronoi bias of the nodes in the tree and breaks this implicit dependence on the boundaries of the environment.

III. Dynamic-Domain RRT Path Planner

The sampling strategy proposed in [28] is based on the notion of *visible Voronoi region*. For a given node of the tree, such region is defined as the intersection of its Voronoi region with the associated visibility domain, i.e. the set of configurations that can be connected to the node by a collision-free local path. While a uniform distribution over such visible Voronoi region (Figure 2 b) would ideally circumvent the bias issue of classical RRTs (Figure 2 a), its computation is however a very hard problem. Another distribution, which retains the good properties of the visibility distribution but is more efficient to compute, is thus proposed (Figure 2 c).

The *visible Voronoi region* of a *boundary* node v, is approximated by its *boundary domain* which is defined as the intersection of the Voronoi region of v and an n-dimensional sphere of radius R centered at v. Then the *dynamic domain* of radius R for a set of points is the boundary domains of the boundary points combined with the Voronoi regions of all other points. The uniform distribution over this domain is called the *dynamic domain distribution*.

In order to obtain the dynamic domain distribution, a distribution from the configuration space is first generated and then restricted to the dynamic domain. Given that the original distribution was uniform, the obtained restriction is also uniform. Computing this dynamic distribution corresponds to the lines 3-6 in the algorithm description of Figure 3. The radius of each point is set to the value R if it is a boundary point and to the value ∞ otherwise. Note, that for this step to be efficient the random configurations are chosen from the area closely fitting the dynamic domain. Practically, it means that we sample inside the smallest bounding box containing all the boundary domains.

The pseudocode of the DD-RRT algorithm is shown on Figure 3. The algorithm updates the information about the boundary points on the fly. At the beginning of the exploration of the tree, all the points are considered to be non-boundary. As soon as the expansion of a given node fails, it becomes a boundary node. This corresponds to the lines 11-12 in the code. The radius field of this node is then updated to R. For the following iterations, the samples from the Voronoi region of this node are restricted to its boundary domain.

Let's remark that the tree tends to produce more nodes in the free space, since the bias of the boundary nodes is reduced. Moreover, many samples may get rejected before the one belonging to the dynamic domain is found. Therefore an efficient nearest neighbor method [2] adapted to the topology of the configuration space should be used.

```
BUILD_DD_RRT(q_init)
 1   T.init(q_init);
 2   for k = 1 to K do
 3       repeat
 4           q_rand ← RANDOM_CONFIG();
 5           q_near ← NEAR_NEIGH(q_rand, T, d_near);
 6       until (d_near < q_near.radius)
 7       if q_new ← CONNECT(T, q_rand, q_near)
 8           q_new.radius = ∞;
 9           T.add_vertex(q_new);
10           T.add_edge(q_near, q_new);
11       else
12           q_near.radius = R;
13   Return T;
```

Fig. 3. The dynamic-domain RRT algorithm with a fixed radius

IV. Influence of the Dynamic-Domain radius

We can distinguish for the RRT based planners two main expansion modes mixed together. The first one is an exploration mode that steers the expansion of the tree toward unexplored regions of the configuration space. This exploration mode is performed through the expansion of the frontier nodes. The second one corresponds to a refinement mode resulting from the addition of new nodes in regions already covered by the tree.

As noted in [28], the value of the dynamic domain radius R should be carefully chosen since it controls the balance between these two expansion modes. The bigger the radius, the stronger the exploration, which also causes the detriment of the tree refinement.

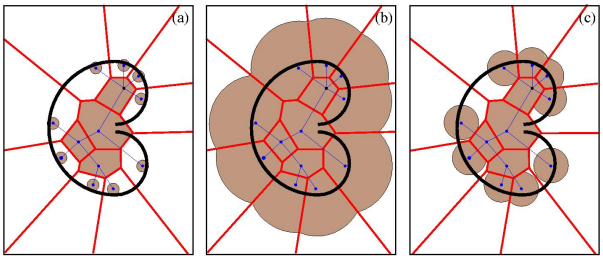


Fig. 4. 3 different radius values for the dynamic domain distribution. In (a), small radius increases the refinement mode of the planner whereas in (b), an important radius provides a bigger weight for the exploration. In (c), refinement and exploration are balanced.

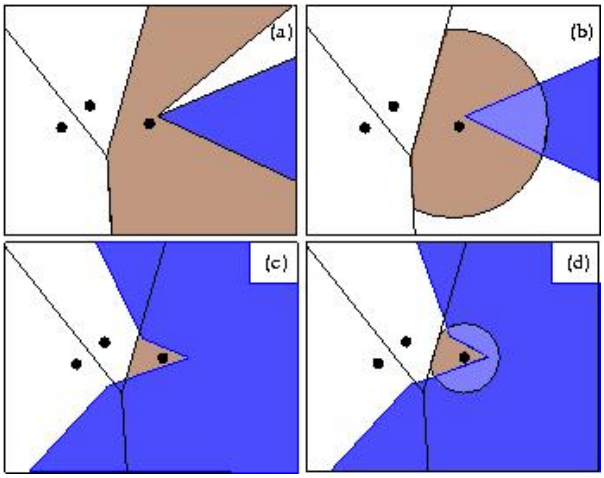


Fig. 5. Visible Voronoi region and associated boundary domain in two situations. In (a and b) the boundary node is only locally close to the obstacle. In (c and d) the boundary node is surrounded by obstacles in almost all the directions.

For a given problem, the balance between these two behaviors can be difficult to find. Too much refinement will add potentially useless nodes at the price of slowing down the planner's performance. On the contrary, a strong bias to exploration can lead to useless attempt to extend frontier nodes that are also boundary nodes, whereas the insertion of a new node in the already explored regions is necessary to discover a new passage and further expand the tree.

In practice, it means that a too small or too important radius decreases the performance of the planner (see Figure 4). This behavior has been experimentally verified on different examples (see graphs of Figures 7 8 and 9). Generally, we can say that the performance remains close to the best one when the value of the radius is set between half and twice the "optimal" radius value. Outside these bounds, the performance rapidly decreases and the further the radius is from the "optimal" one, the worse the performance is.

In the initial DD-RRT algorithm, the radius of each point can only have two values : the value R if it is a boundary point and ∞ otherwise. There is no transition between these two values. As we previously explained, we want the boundary domain to fit as much as possible the visible Voronoi region of the point but this domain is very different depending on whether it is strongly surrounded by obstacles or not (see Figure 5). Nevertheless, as long as the node is defined as a boundary node, its bounding domain is always defined by the same given radius.

In the following, we propose to adapt the radius of each of the nodes independently during the search process. This adaptation has two main advantages. First it helps to automatically balance the weight allocated to exploration and refinement. Secondly it allows to differentiate locally the behavior of the planner and thus better adapt it to the shape of the free space.

V. Adaptive tuning of the Sampling Domain

BUILD_ADAPTIVE_DD_RRT(q_{init})
1 $\mathcal{T}.init(q_{init})$;
2 **for** $k = 1$ **to** K **do**
3 **repeat**
4 $q_{rand} \leftarrow$ RANDOM_CONFIG();
5 $q_{near} \leftarrow$ NEAR_NEIGH($q_{rand}, \mathcal{T}, d_{near}$);
6 **until** ($d_{near} < q_{near}$.radius)
7 **if** $q_{new} \leftarrow$ CONNECT($\mathcal{T}, q_{rand}, q_{near}$)
8 q_{new}.radius = ∞;
9 **if** q_{near}.radius $\neq \infty$;
10 q_{near}.radius = $(1+\alpha) \times q_{near}.radius$;
11 $\mathcal{T}$.add_vertex(q_{new});
12 $\mathcal{T}$.add_edge(q_{near}, q_{new});
13 **else**
14 **if** q_{near}.radius = ∞
15 q_{near}.radius = R;
16 **else**
17 q_{near}.radius = $(1-\alpha) \times q_{near}.radius$;
18 Return $\mathcal{T}$;

Fig. 6. The dynamic-domain RRT algorithm with an adaptive radius.

The example of Figure 5 illustrates two kinds of situations for a boundary node. In the first one (a and b), despite the proximity to the obstacle, the visible Voronoi region of the node remains a large region (a) and would be better approximated by a boundary domain with a big radius value (b). In the second case (c and d) the node is strongly surrounded by obstacles and thus its visible Voronoi region is small (c). A boundary domain with a small radius value (d) is better suited for this case.

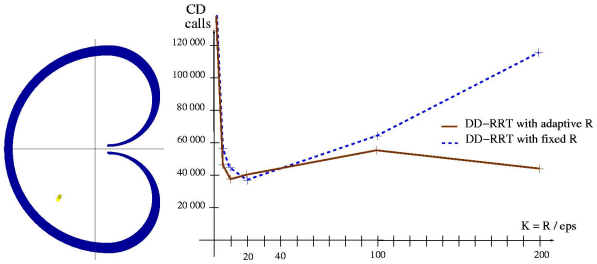
This observation means that the information gained during the various attempts to extend a node can help to have a better evaluation of the visible Voronoi domain surrounding the node. Each time the expansion of a given node fails increases the probability of the node to be strongly surrounded by obstacles. Each time it successes, this probability decreases. Consequently, we propose to adapt the boundary domain of a given node (i.e. its associated radius) as a function of the number of expansion attempts and failures from this node.

The pseudocode representing the DD-RRT algorithm including the adaptive radius modification is shown on Figure 6. As soon as the expansion from one of the nodes fails, the node becomes a boundary node. Its radius is then initialized to a given value. Then, each time the node is chosen for

expansion, its associated radius is modified depending on the success or the failure of the expansion attempt. If the attempt succeeds and then leads to the creation of a new node in the tree the radius value is increased by a given percentage α (line 10 of the algorithm). On the contrary, if the expansion fails the radius value is decreased by the same ratio (line 17 of the algorithm).

To keep the probabilistic completeness of the algorithm we always ensure the possibility for a node to be extended. To do so, we put a lower bound on the possible radius values of the nodes (not shown in the algorithm of Figure 6). This bound is a multiple of the interpolation step used to check the collisions when we attempt to expand a node.

A last point is to define a value for α. Experimentally, we have noticed that the value of this parameter is not at all as critic as the radius parameter was in the initial version of DD-RRT. A small value for α (usually a few percents) is sufficient to increase the robustness of the algorithm and is used in the experiments presented in the next section.



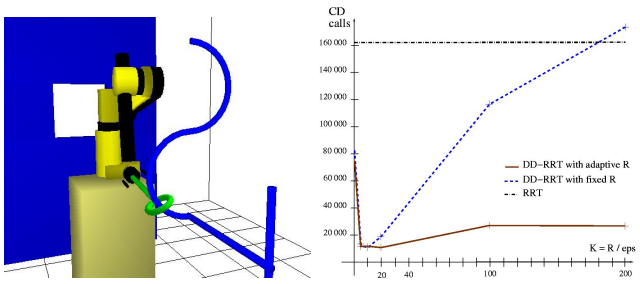
	Adaptive radius					RRT
K	5	10	20	100	200	
time (s)	14	12	14	28	38	719
no. nodes	6 263	4 992	5 134	5 929	4 909	6 929
CD calls	47 440	38 711	40 869	51 300	44 832	1 627 974

	Fixed radius				
K	5	10	20	100	200
time (s)	17	15	13	28	66
no. nodes	7 522	5 868	4 665	4 638	5 136
CD calls	56 900	45 596	38 229	59 384	115 214

Fig. 7. Results on a bug trap environment. The robot is a 2dof cylinder shown in the bottom left inside the bug trap.

VI. Experimental Results

The algorithms were implemented in C with the software platform Move3D developed at LAAS [25]. The experiments were performed on a 333 MHz Sunblade 100 running SunOs 5.9 and compiled under gcc 3.3. For each example the performances for the mono-directional versions of three algorithms, the classic RRT, DD-RRT with fixed radius and our new version of DD-RRT with an adaptive radius for each node, are compared. The initial radius values R are defined as a multiple of the interpolation step ϵ, $R = K\epsilon$. Graphs of Figures 7, 8 and 9 show the evolution of the performance (in term of collision detection calls) as a function of this K parameter. Tables indicate for each experiment and each value of K the running times, the number of nodes in the solution trees and the number of collision detection (CD) calls during the construction process. All the reported values are averaged over 50 runs. We first show (Figure 7) the results obtained



	Adaptive radius					RRT
K	5	10	20	100	200	
time (s)	34	24	21	75	80	240
no. nodes	1 029	1 009	718	943	844	1 314
CD calls	11 166	11 160	11 468	27 786	27 343	162 570

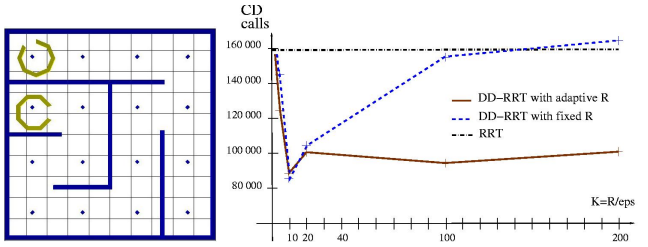
	Fixed radius				
K	5	10	20	100	200
time (s)	47	21	28	166	245
no. nodes	1 277	890	724	809	856
CD calls	12 408	10 871	18 107	117 903	174 714

Fig. 8. Results for a 6dof manipulator arm. the goal is to extract the hoop-like object from the S-shaped bar.

for the 2 dimensional bug trap. Although it does not appear in the figure, the size of the environment was chosen to be much larger (150 times) than the volume occupied by the bug trap.

For the original DD-RRT the optimal radius tuning is close to $R = 20\epsilon$. For this radius value the number of collision detection calls is more than 40 times larger than for the classic RRT planner! When the value of the radius is smaller than the optimal one, the performance decreases very quickly. When it is larger, it decreases slower, almost linearly as a function of the radius. With the adaptive tuning variant the performance is also affected when the radius value is smaller than the optimal, but quite stable when it is larger. We can see that this robustness holds even for a radius 10 times bigger than the initial one.

Note that for the bug trap example, all the boundary nodes have almost the same kind of "contact" region with the obstacles. It means that in average, the evolution of the radius for all the boundary nodes is probably quite similar. Thus, it may indicate that the good performance of the adaptive method is not only due to the local differentiation of the radius attributed to each node but comes also from a global good balancing between exploration and refinement.



	Adaptive radius					RRT
K	5	10	20	100	200	
time (s)	206	157	179	148	144	96
no. nodes	22 190	11 932	12 160	9 186	9 783	10 419
CD calls	125 233	88 781	103 512	94 139	102 609	159 392

	Fixed radius				
K	5	10	20	100	200
time (s)	124	58	68	93	95
no. nodes	22 685	9 604	9 503	10 524	10 678
CD calls	144 197	86 482	104 953	154 201	165 035

Fig. 9. Results for a maze environment. The C-shaped robot has to move from one extremity of the maze to the other by combining rotations and translations.

Next experiment involves a constrained motion planning problem for a 6 dof manipulator arm carrying a hoop-like object which has to be extracted from a S-shaped bar. For a fixed radius, the efficiency of the DD-RRT algorithm decreases very quickly when the gap between the radius set and the optimal one increases. When this radius is 10 times bigger than the optimal one ($K = 100$ instead of 10), the gain in terms of collision detection calls becomes less than 40 percents. For the adaptive DD-RRT this gain is still a factor of 6. Also note, that with the adaptive tuning the performance still decreases but very slightly from the initial setting.

Last example in Figure 9 is a 2 dimensional labyrinth which combines 2d rotations and translations (it gives 3 degrees of freedom in the configuration space). The goal is to move the C-shaped robot from one corner of the labyrinth to another. The collision checks are very cheap in this problem. It explains the fact that the average time to solve the problem is bigger when we use the adaptive version of the algorithm. Nevertheless, the number of collision detection calls is still very controlled when we use this version.

As we can see, the performance robustness of the algorithm relatively to the radius parameter is in general largely increased in our adaptive version of the algorithm. The gain is particularly significant when the initial value is an overestimation of the optimal value compared to an under estimation. It means that with its low sensitivity to the parameter the adaptive tuning method allows to initialize the planner with an important radius without any significant lost in performance.

VII. Conclusions and Future Work

The DD-RRT algorithm is an algorithm especially efficient for problems with complex geometries where the collision tests are expensive. In this work we have proposed a new extension of the dynamic-domain RRT algorithm. This extension significantly increases the robustness of the planner by automatically adapting the region of influence of each node during the search process. Several other directions of research remain to improve the DD-RRT algorithm.

In particular, the information about the distance to the obstacles could be used to define the dynamic domain.

Another important direction is to investigate the application of the DD-RRT framework to other constrained motion planning problems such as planning for closed linkages or planning under differential constrains, where both the cost of the iteration computations and the Voronoi bias greatly affect the efficiency of planning algorithms.

References

[1] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE Int. Conf. Robot. & Autom.*, pages 113–120, 1996.

[2] A. Atramentov and S. M. LaValle. Efficient nearest neighbor searching for motion planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 632–637, 2002.

[3] R. Bohlin and L. Kavraki. Path planning using Lazy PRM. In *IEEE Int. Conf. Robot. & Autom.*, 2000.

[4] V. Boor, N. H. Overmars, and A. F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *IEEE Int. Conf. Robot. & Autom.*, pages 1018–1023, 1999.

[5] P. Choudhury and K. Lynch. Trajectory planning for second-order underactuated mechanical systems in presence of obstacles. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, 2002.

[6] J. Cortés and T. Siméon. Sampling-based motion planning under kinematic loop-closure constraints. In *6th International Workshop on Algorithmic Foundations of Robotics*, pages 59–74, 2004.

[7] E. Ferré and J.-P. Laumond. An iterative diffusion method for part disassembly. In *IEEE Int. Conf. Robot. & Autom.*, 2004.

[8] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance and Control*, 25(1):116–129, 2002.

[9] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *IEEE Int. Conf. Robot. & Autom.*, pages 1718–1723, May 1990.

[10] L. Han and N. M. Amato. A kinematics-based probabilistic roadmap method for closed chain systems. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 233–246. A K Peters, Wellesley, MA, 2001.

[11] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *IEEE Int. Conf. Robot. & Autom.*, pages 1408–1413, 2000.

[12] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In et al. P. Agarwal, editor, *Robotics: The Algorithmic Perspective*, pages 141–154. A.K. Peters, Wellesley, MA, 1998.

[13] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. Comput. Geom. & Appl.*, 4:495–512, 1999.

[14] S. Kagami, J. Kuffner, K. Nishiwaki, and K. Okada M. Inaba. Humanoid arm motion planning using stereo vision and RRT search. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2003.

[15] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. Planning collision-free reaching motions for interative object manipulation and grasping. *Eurographics*, 22(3), 2003.

[16] 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.

[17] J. Kim and J. P. Ostrowski. Motion planning of aerial robot using rapidly-exploring random trees with dynamic constraints. In *IEEE Int. Conf. Robot. & Autom.*, 2003.

[18] F. Lamiraux, E. Ferre, and E. Vallee. Kinodynamic motion planning: connecting exploration trees using trajectory optimization methods. In *IEEE Int. Conf. Robot. & Autom.*, pages 3987–3992, 2004.

[19] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

[20] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Wellesley, MA, 2001.

[21] E. Mazer, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. Artificial Intell. Res.*, 9:295–316, November 1998.

[22] G. Sánchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Int. Symp. Robotics Research*, 2001.

[23] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. H. Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *Int. J. Robot. Res.*, 17:840–857, 1998.

[24] T. Siméon, J.-P. Laumond., and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6), 2000.

[25] T. Siméon, J.P. Laumond, C. van Geem, and J. Cortés. Computer aided motion: Move3d within molog. In *Proc. of IEEE Int. Conf. Robotics and Automation*, 2001.

[26] P. Svestka and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *IEEE Int. Conf. Robot. & Autom.*, pages 1631–1636, 1995.

[27] J. Yakey, S. M. LaValle, and L. E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17(6):951–958, December 2001.

[28] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. In *Proc. of IEEE Int. Conf. Robotics and Automation, to appear*, Barcelona, Spain, 2005.