Proceedings of the 2006 IEEE International Conference on Robotics and Automation
Orlando, Florida - May 2006

An Obstacle-Based Rapidly-Exploring Random Tree

Samuel Rodríguez† sor8786@cs.tamu.edu
Xinyu Tang† xinyut@cs.tamu.edu
Jyh-Ming Lien† neilien@cs.tamu.edu
Nancy M. Amato† amato@cs.tamu.edu

***Abstract*— Tree-based path planners have been shown to be well suited to solve various high dimensional motion planning problems. Here we present a variant of the Rapidly-Exploring Random Tree (RRT) path planning algorithm that is able to explore narrow passages or difficult areas more effectively. We show that both workspace obstacle information and C-space information can be used when deciding which direction to grow. The method includes many ways to grow the tree, some taking into account the obstacles in the environment. This planner works best in difficult areas when planning for free flying rigid or articulated robots. Indeed, whereas the standard RRT can face difficulties planning in a narrow passage, the tree based planner presented here works best in these areas.**

I. Introduction

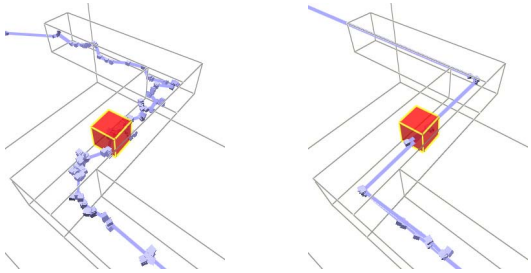
Algorithmic motion planning studies the problem of finding a sequence of feasible movements for a movable object (robot) to maneuver among obstacles from a start configuration to a desired goal configuration. Motion planning problems are important and have many applications beyond robotics including games, virtual surgery and CAD [4], [8], [16]. A motion planning problem is usually given as a (geometric) description of the world (called workspace), which normally consists of a robot and a set of obstacles, e.g., the workspace shown in Figure 4.

Many techniques have been developed to solve this problem. Most of these methods translate the problem described in the workspace into the configuration space (C-space) or the set of all possible configurations of the robot. Paths are sequences of consecutive points in C-free connecting start and goal configurations. Unfortunately, it is believed that any complete motion planner (one that will find a path if one exists or report that none exists otherwise) will have exponential time complexity with respect to the number of degrees of freedom of the robot [21]. Due to this reason, attention has shifted to randomized planning methods.

Probabilistic Roadmap Methods (PRMs) [13] are one of the most popular types of sampling-based planners. PRMs construct a roadmap by first randomly generating feasible configurations and then connecting these configurations using simple local planners. Finally, a solution can be extracted by connecting both start and goal configurations to the resulting roadmap. Tree-based planners, another common type of sampling-based planner, explore the space from a root configuration by adding more configurations to the existing tree.

*This research supported in part by NSF Grants EIA-0103742, ACR-0081510, ACR-0113971, CCR-0113974, ACI-0326350, and by the DOE. Rodriguez supported in part by a National Physical Sciences Consortium Fellowship.

†Parasol Lab., CS Dept., Texas A&M University.



(a) (b)

Fig. 1. Differences when growing a tree: (a) basic expansion and (b) using obstacle information.

Issues. A problem arises when there are difficult areas of the configuration space to explore. There have been many attempts to generate nodes in these difficult or interesting areas [1], [7], [22] or to classify certain areas of the configuration space as narrow or cluttered regions [17]. Some PRM-based planners have been shown to be more effective at finding these areas.

There are still problems when trying to explore these difficult areas once they have been identified. Tree-based planners that explore in an unbiased manner, such as RRT, have difficulty growing out of narrow passages. In narrow or tight areas, we will show how tree-based planners can use local information to more effectively explore these regions, see Figure 1 for an example.

Our approach. We will show that some hints can be obtained which can be helpful in growing the tree. These hints can be obtained from tessellation of workspace obstacle surfaces or approximate C-obstacle surfaces. We will show that, with a tree based planner, using the shape of the obstacle to steer the direction that the tree should grow can be particularly useful when exploring difficult areas of the configuration space. For instance, triangles representing the obstacle surface(s), such as those shown in Figure 4, can provide useful information. Also, we show that other ways to expand a tree can help when using tree-based path planning methods. The resulting planner is faster and consistently solves problems in fewer iterations. In our results, we compare the planners' abilities to explore difficult regions (which is different from testing their abilities to find these difficult areas).

II. Related Work

Sampling-based algorithms have been widely used in solving motion planning problems. A randomized potential field method, RPP [3], works very well when the C-space is relatively uncluttered, but unfortunately there also exist simple situations in which they are not successful [12]. There are also

other probabilistic methods such as the *Probabilistic Roadmap Methods* (PRMs) mentioned above, which have been shown to perform well in a number of practical situations, see, e.g., [13].

Tree-based planners. Other randomizied methods for path planning incrementally construct trees that explore a connected region of C-space. The Ariadne's Clew algorithm [6] explores the free space using "Explore" and "Search" algorithms to build and connect landmarks. A tree-based path planner was developed by LaValle and Kuffner in [14] and [15] that is useful for exploring C-space. This path planning technique is known as the Rapidly-Exploring Random Tree (RRT). A similar tree-based planner, Expansive Space Tree (EST), was developed by Hsu *et. al* in [11]. By only exploring the relevant portion of configuration space needed for the query, it works well for single query problems. Both ESTs and the RRT-Connect method [14] have the ability to bias two trees toward each other. As a variant of RRT, Dynamic Domain RRT [23] selects the sampling region based on the visibility region of the nodes in the tree for a better exploration.

The Sampling-based Roadmap of Trees (SRT) [19] algorithm combines PRMs and sampling-based tree methods. It first uses PRMs to sample random configurations as milestones and then grows trees from each node.

Planners that tackle narrow passage problems. Many methods have been proposed to generate nodes in difficult areas, called *narrow passages* in C-space [10]. A number of methods specifically targeted at this problem have been proposed, e.g., [1], [7], [9], [10], [22].

Some methods are designed to find narrow passages. In [5], narrow passages in the workspace are found and sampled more densely. This method works well when the workspace is similar to the configuration space. Morales et al. [17] propose a method where certain types of areas of the configuration space were identified, such as narrow or cluttered regions.

The methods mentioned above are effective in finding interesting or difficult areas. These are often the areas that need to be explored more in order to solve a given problem, which usually include narrow or clutter areas. In a complete framework, these difficult areas would be where we would apply our Obstacle-Based Rapidly-Exploring Random Tree (OBRRT).

Planners that use workspace hints. Some planners use the known workspace information to help sample configurations. In [1], [18], [7], nodes are added near the boundaries of the obstacles in the workspace. Redon and Lin also propose a local planning method in contact space [20]. In [5], narrow passages in the workspace are identified and sampled more densely.

III. Preliminaries

In this section, we will define terms and notation that we use in this paper.

Basic RRT. The basic RRT-expand algorithm presented in [15] is shown in Algorithm 1. The tree grows from the nearest neighbor in the tree, x_{near}, toward a configuration that was randomly generated, x_{rand}. The step length described in [15] was to be some small distance but in [14] a greedy approach was described such that larger steps can be taken. As will be shown in Section VI, the basic tree-based expansion can be improved.

Algorithm 1 Basic RRT expansion method.

Require: tree $\mathcal{T}$ and Iterations K
1: **for** $i = 1 \ldots K$ **do**
2: x_{rand} = random configuration
3: x_{near} = nearest neighbor in tree $\mathcal{T}$ to x_{rand}
4: x_{new} = extend x_{near} toward x_{rand} for step length
5: **if** (x_{new} can connect to x_{near} along valid edge) **then**
6: $\mathcal{T}$.AddVertex(x_{new}), $\mathcal{T}$.AddEdge(x_{new}, x_{near})
7: **end if**
8: **end for**
9: **return** $\mathcal{T}$

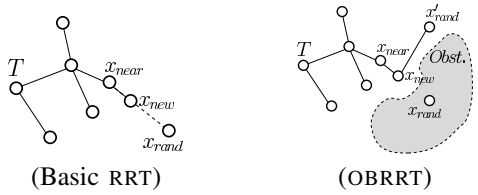


Fig. 2. A comparison of Basic RRT and Obstacle-Based RRT (OBRRT)

Obstacle Vectors. In our planner, we use obstacle vectors obtained from the obstacle. The vectors are used to assist in selecting a direction for the tree to try and grow. Vectors obtained from a 3-dimensional obstacle are taken from the triangles (as that is how we model our environments) that describe the obstacle. Given some triangle, a direction for the tree to grow can be obtained from the vertices of the triangle.

A triangle in three dimensions is described by three vertices, a, b, and c, in the three dimensional space. The directions obtained from the triangle can be chosen randomly from any of the following six directions: $\pm(a-b)$,$\pm(a-c)$, and $\pm(b-c)$, as shown in Figure 3.

Perturbing each component of the obstacle vector by some small value, δ, can be useful when deciding on the direction. In this way, as the orientation of the robot changes near the obstacle surface, the robot does not immediately collide with the obstacle.

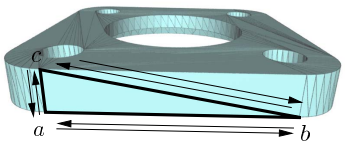


Fig. 3. Possible obstacle vectors obtained from a triangle before perturbing.

IV. A Tree-Based Planner taking Obstacle Hints

Although a Rapidly-Exploring Random Tree does a good job of exploring many areas, there are many modifications that can be made to improve how the method explores narrow or difficult areas of C-space. Current tree-based planners explore difficult areas in the same way that free areas are explored. By only growing the tree toward configurations that are randomly generated, the probability of finding a path out

of the difficult area is greatly reduced. We will show that using the modifications presented here will result in a method that performs much better than the standard RRT-expansion algorithm, especially in difficult areas of the environment.

A. Purpose

The most important modifications that can be made are how the tree decides to grow. In the standard algorithm, the configuration chosen to grow towards is completely random. Although this is a useful way to explore an area, it should not be the only way to grow the tree in difficult areas and could result in very little expansion in these areas. A number of ways to grow the tree will be described which will result in a tree that explores difficult or narrow areas faster and in fewer iterations than in the standard way.

B. Algorithm

Growing a tree when possibly taking obstacle hints can be seen as a four step process: 1) Choose a (source) node to grow from, x_{near}, 2) Choose a growth method, G_i, 3) Generate target configuration x'_{rand}, and 4) Extend from source configuration x_{near} toward target configuration x'_{rand}.

Although in Step 1, there could be many ways to choose nodes as possible nodes for expansion, we choose nodes in the same way a node is chosen in [15], in which a configuration is generated at random and the nearest neighbor in the tree is found. This will be the node that will be expanded. Step 2 and 3, on choosing the way to grow and generating the target configuration will be discussed in Section V and Step 4 will be discussed in Section V-B.

The whole algorithm can be seen in Algorithm 2. The important modifications to the algorithm are in line 5 where the target node is generated and in lines 6–9 where x_{near} is extended toward x'_{rand} taking the greedy approach described later.

Algorithm 2 Obstacle-Based RRT expansion method.

Require: tree $\mathcal{T}$ and Iterations K
1: **for** $i = 1 \ldots K$ **do**
2: x_{rand} = random configuration
3: x_{near} = nearest neighbor in tree $\mathcal{T}$ to x_{rand}
4: G_i = Select a Growth Method
5: x'_{rand} = GenerateTargetNode(x_{near}, x_{rand}, G_i)
6: x_{new} = extend(greedy) x_{near} toward x'_{rand}
7: **if** (x_{new} can connect to x_{near} along valid edge) **then**
8: $\mathcal{T}$.AddVertex(x_{new}), $\mathcal{T}$.AddEdge(x_{new}, x_{near})
9: **end if**
10: **end for**
11: **return** $\mathcal{T}$

V. Possible Ways To Expand

In this section, we consider how the tree can be expanded. The description of the obstacle can be useful when deciding which direction to grow. Nine possible ways to expand a tree have been identified, although there are other ways that can be used. In this section we will describe how x'_{rand} is generated and how the tree can be expanded by each growth method.

A. Growth Methods

In this section, we describe various growth methods. In each growth method, the orientations used in the ways to grow are either the same as the source configuration or random orientations. Random orientations are useful when complex movements of the robot are required to move through narrow or difficult areas. Fixed orientations are most useful when a sliding motion of the robot is needed to move through a narrow or difficult area or when only simple translation is needed.

G_0**: Basic Extension.** This is the standard way of expanding a tree toward a random configuration. In this growth method, x'_{rand} is set to x_{rand}.

G_1**: Random position, Same orientation.** In this way of growing, x'_{rand} is obtained by setting the the translational degrees of freedom to a random position and keeping the orientational degrees of freedom from x_{near}. Hence, this method only uses translation.

G_2**: Random obstacle vector, Random orientation.** In this growth method, $x'_{rand} = x_{near} + \overrightarrow{OV}$. In this case $\overrightarrow{OV}$ is a random obstacle vector from all of the triangles in the environment. The orientational degrees of freedom of x'_{rand} are set randomly.

G_3**: Random obstacle vector, Same orientation.** The target configuration, x'_{rand}, is obtained as in G_2 except the orientational degrees of freedom are set up for translation only. In this way the orientation of x'_{rand} is the same as x_{near}.

G_4**: Rotation followed by Extension.** In this way of growing the source configuration is first rotated to align with the target configuration until it is aligned or there is collision. It is then extended toward the target configuration until collision or the target configuration, x_{rand}, is reached. This can be seen as growing with a modified rotate-at-s local planner where s = 0 [2]. Growing toward x'_{rand} can be seen as extending from x_{near} to x_{rand1}, where x_{rand1} is only a change in orientation followed by a transition from x_{rand1} to x_{rand}.

G_5**: Trace obstacle, Random Orientation.** In this way of growing, the first colliding triangle is found after growing by G_0. An obstacle vector is obtained from the first detected obstacle triangle that caused the collision. The source configuration is extended in that direction with a randomly generated orientation. This is useful when growing in areas where difficult movements are needed.

G_6**: Trace obstacle, Same Orientation.** This way of growing is the same as G_5 but the orientation from x_{near} is used rather than a random orientation. This is useful when a sliding motion is needed.

G_7**: Trace C-space obstacle.** In this growth method, we try to find a vector parallel to a C-Space obstacle boundary. It is not feasible to compute the actual C-Space boundary, so here we approximate it. We first shoot a small number of rays (our current results use two) within a gaussian distance *d*, of each other, then find their collision configurations x_{col1} and x_{col2}. The C-obstacle vector is calculated as the vector connecting the colliding configurations: $\overrightarrow{COV} = x_{col2} - x_{col1}$. In this way $x'_{rand} = x_{near} + \overrightarrow{COV}$.

G_8**: Medial Axis Push.** Same as in G_5 but after the target configuration has been pushed in the obstacle direction, it is

then pushed toward the medial axis of the configuration space. In this way x_{near} is extended toward x'_{rand} which is near the medial axis.

The random obstacle vectors in G_2 and G_3 are randomly chosen from all of the triangles describing the obstacles in the environment. The random obstacle vectors used in G_5, G_6, and G_8 are obtained from the triangle that last caused collision. C-obstacle vectors are obtained directly from two colliding configurations approximating the tangent of the C-obstacles.

B. Greedy Modification

In [15], the step length that the tree would grow towards the random configuration is small. An important improvement would be to make this a greedy algorithm such that it would take as big a step length as possible, as long as it is less than some maximum step length specified. Also, instead of getting as close to the obstacle as possible, some distance can be specified such that the final configuration will have some space between itself and the obstacle. In this way, if that node is chosen as a source configuration for the tree to grow from, it will be easier to extend that node.

VI. Experimental Setup and Results

In this section, we compare our Obstacle-Based RRT (OBRRT) to the Basic RRT and Greedy RRT expansion methods. OBRRT is tested using two variations OBRRT$_{tuned}$ and OBRRT$_{avg}$. Using OBRRT$_{tuned}$, parameters are specified to be well suited for exploring each environment. OBRRT$_{avg}$ assigns the same weight for each growth method. The basic and greedy RRT will differ in maximum possible step size. The basic RRT will have a small step size as described in [15]. The greedy RRT will take as large a step size as possible leaving some space between the robot configuration and the obstacle as described in Section V-B. OBRRT will utilize the various modifications described. The code is compiled under gcc 3.3. All tests are on an AMD Athlon XP 2800+ processor with 768 MB of RAM.

The environments tested are the s-tunnel environment (medium, long and large robot), flange environment (scale 0.90, 0.95 and 1.0), OBRRT-tunnel environment with an articulated robot and the maze environment, also with an articulated robot. Each method, OBRRT, Greedy RRT, and Basic RRT are tested 10 times for each environment and the results shown represent the average of all the test runs.

The results reported will include the average number of iterations, collision detection calls (CDs), nodes required, and time needed to grow the tree such that the query was satisfied. It is important to note the time and number of iterations required as this gives an idea as to how long each method can be expected to grow out of the difficult regions.

A. Selecting a Tree Root

All environments are tested from an initial predefined root configuration in the environment. At each test iteration, a tree is expanded from the same initial configuration for a given number of iterations. The number of iterations is increased until the query is satisfied. Although we predefine this root or start configuration for testing purposes, any node or connected component could be used in a general framework from where to start expanding.

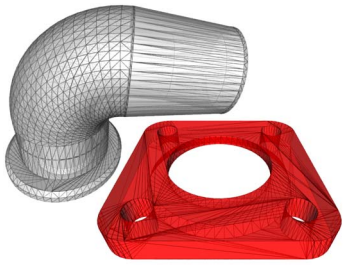


Fig. 4. In this workspace, both obstacle and robot are represented as triangulated meshes. For this environment, the curved tube-like object (the robot) should fit in the gap in the obstacle (largest hole).

B. Flange Environment

TABLE I

RESULTS FOR FLANGE SCALE(0.90, 0.95 AND 1.0) ENVIRONMENT.

flange, scale 0.90				
Method	Time	Nodes	Iterations	CDs
OBRRT$_{tuned}$	37.64	226.6	1750	12122.2
OBRRT$_{avg}$	65.94	227.7	2660	62759.3
Greedy RRT	73.66	109.4	4440	22499.8
Basic RRT	67.56	91.4	4160	19457.9
flange, scale 0.95				
Method	Time	Nodes	Iterations	CDs
OBRRT$_{tuned}$	143.19	479.8	3220	28019.7
OBRRT$_{avg}$	278.52	589.2	5050	103177.3
Greedy RRT	328.79	109.2	13260	48262.9
Basic RRT	406.56	118.4	15250	50571.8
flange, scale 1.0				
Method	Time	Nodes	Iterations	CDs
OBRRT$_{tuned}$	227.06	1011	6100	362768.0

In the flange environment, the curved tube shown in Figure 4 should slide out of the gap in the obstacle. The scaled environments have been scaled down from the original version by a factor of 0.90 and 0.95. The results for this environment are shown in Table I. The initial configuration that these trees expand from is a configuration where the flange is already in the obstacle. The tree grows until it can find a path where the two objects were separated, as shown in Figure 4.

In both of the tests, flange scale 0.9 and 0.95, OBRRT$_{tuned}$ is able to solve the query much faster and in fewer iterations and collision detection calls than the other methods tested. For the flange, scale 0.95, OBRRT$_{tuned}$ is able to solve the problem close to twice as fast as any of the other methods. OBRRT$_{avg}$ is also able to solve these environments in fewer iterations and somewhat faster than Basic and Greedy RRT. In both cases OBRRT is on average faster at solving the queries than Basic or Greedy RRT. The number of nodes needed by both versions of OBRRT are more than the Basic and Greedy RRT but this could be because the space is searched near the obstacle surface resulting in some expansion that is only incremental.

To show the effectiveness of OBRRT, we were able to solve the original flange environment (scale 1.0) nearly as fast as OBRRT was able to solve the version scaled 0.95. Basic and

Greedy RRT were unable to find a solution given 100,000 iterations.

C. S-Tunnel Environment

In the s-tunnel environment, a configuration is placed in the middle of the passage as the root configuration. The goal configurations for the query are on opposite sides of the environment. In this way, the tree will grow until the tree exits both ends of the tunnel.

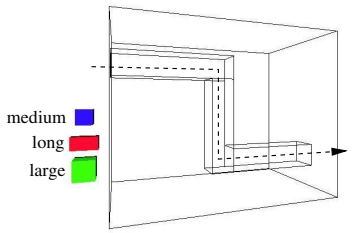


Fig. 5. s-tunnel environment. This environment has a curved tunnel through which the robot should pass through. The robots, medium, long and large, are displayed from top to bottom.

TABLE II
RESULTS FOR S-TUNNEL (MEDIUM,LONG,LARGE) ENVIRONMENT.

s-tunnel, medium robot, Difficulty: easy				
Method	Time	Nodes	Iterations	CDs
OBRRT$_{tuned}$	0.83	190.9	197	18618.2
OBRRT$_{avg}$	1.44	213.6	235.2	31823.3
Greedy RRT	15.35	541.4	3780	71685.5
Basic RRT	15.34	518.1	3390	66681.7
s-tunnel, long robot, Difficulty: medium				
Method	Time	Nodes	Iterations	CDs
OBRRT$_{tuned}$	2.81	589.1	699.3	29739.2
OBRRT$_{avg}$	7.65	773.4	979.9	113796.2
Greedy RRT	212.58	1311.70	11322	190210.0
Basic RRT	824.76	1963.7	13022	283595.8
s-tunnel, large robot, Difficulty: hard				
Method	Time	Nodes	Iterations	CDs
OBRRT$_{tuned}$	1.01	117.6	331	6769.4
OBRRT$_{avg}$	2.05	215.5	287.2	37231.5
Greedy RRT	699.60	2740.6	40513	468453.9
Basic RRT	2162.04	3582.5	51500	615989.5

The s-tunnel environment was tested with three different robots. The three robots can be seen in Figure 5. The medium sized robot is the easiest when exploring the narrow passage. The long and large robot are more difficult to plan for. In planning for the long robot, when the tunnel changes directions, the robot has to rotate and make a complicated series of moves to get out of the tunnel. The large robot has only a minimal amount of space to move in the tunnel making this a difficult problem for most planners. For the large robot, the robot has to slide through the narrow passage.

The results for these environment for each type of robot can be seen in Table II. With each type of robot, OBRRT is able to easily find a path out of the passage and performs much better than the other methods in each feature. Exploring this passage is difficult for Greedy or Basic RRT since exploring the passage can only be done in very specific directions. These directions are easily obtained using OBRRT. In each case OBRRT$_{tuned}$ outperforms OBRRT$_{avg}$, although only slightly.

For each robot type, OBRRT is able to find a path much quicker and with far fewer collision detection calls than the other two methods. In fact for the large robot, Basic RRT requires many more collision detection calls and takes much longer than either OBRRT or Greedy RRT. Since both the Greedy RRT and OBRRT leave some space between the new robot configuration generated and the obstacle, expanding could be somewhat easier. A similar result is seen for each version of the robot used showing the difficulty both Basic and Greedy RRT can have when searching a narrow or difficult area.

D. OBRRT-*tunnel Environment*

A 3-link, 8 DOF, articulated robot is placed at a configuration inside the first "R" in the OBRRT-tunnel environment. The goal configruations are at either end of the tunnel, as shown in Figure 6. The tree is expanded until it can connect to each of the goal configurations and so, has to find a path through the letters.

As shown in Table III, OBRRT$_{tuned}$ is able to explore this environment more effectively than OBRRT$_{avg}$. This can be seen in each of the features: time, nodes, iterations and collision detection calls. It should be noted that neither Basic or Greedy RRT were able to expand out of the letters tunnel given up to 50,000 iterations.

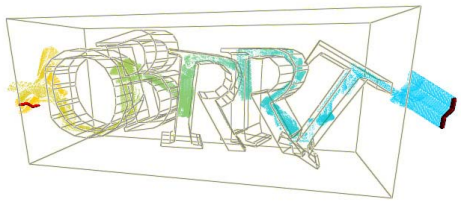


Fig. 6. OBRRT-tunnel environment with an articulated robot.

TABLE III
RESULTS FOR OBRRT-TUNNEL ENVIRONMENT.

Letters, articulated robot, 3 links, 8 DOF				
Method	Time	Nodes	Iterations	CDs
OBRRT$_{tuned}$	2730	6225	10500	4488508
OBRRT$_{avg}$	6442	7501	13000	8749241

E. Maze Environment

In this environment a 4-link articulated robot is placed inside the maze. Goal configurations are placed on both ends of the maze as shown in Figure 7. The tree is expanded until it can connect to each of the goal configurations.

From Table IV, it is clear that OBRRT$_{tuned}$ is able to explore this environment more effectively than the other methods tested. This is can been seen in that it is more than twice as fast as any other method and requires fewer nodes, iterations and collision detection calls. OBRRT$_{tuned}$ is shown to be more reliable in that is can quickly grow out of these difficult set of

tunnels to reach the goal configurations. Although OBRRT$_{avg}$ is slower than the other methods, it consistently solves the problem with fewer iterations than Basic and Greedy RRT.

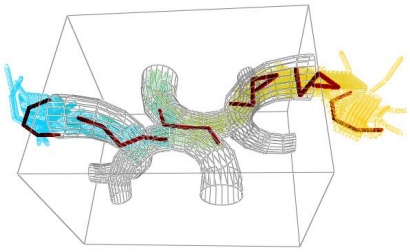


Fig. 7. Maze environment with an 4-link articulated robot.

TABLE IV
RESULTS FOR MAZE ENVIRONMENT.

Maze, articulated robot, 4-links, 9 DOF				
Method	Time	Nodes	Iterations	CDs
OBRRT$_{tuned}$	295	2428	7800	355705
OBRRT$_{avg}$	1089	4452	10400	1708887
Greedy RRT	710	3232	25000	936264
Basic RRT	965	4458	25950	1351933

VII. DISCUSSION

Using a variety of ways to grow a tree can be very beneficial when exploring difficult or narrow areas. Some of the ways proposed to grow the tree involve taking into account the description of the obstacle(s) in the workspace. The ways to grow that take into account the description of the obstacle are some of the most useful ways to grow that are proposed.

From the results, it is clear that OBRRT$_{tuned}$ performs better than the basic and greedy RRT algorithms which in most cases is also true for OBRRT$_{avg}$. This can be seen in the time to make the tree, number of collision detection calls and iterations required to make the tree. We have had success solving the alpha puzzle and flange environment, both of scale 1.0, using OBRRT$_{tuned}$ although OBRRT$_{avg}$ has been less successful for these problems.

There is an issue when assigning weights for each growth method for OBRRT$_{tuned}$. We have performed some preliminary work on being able to adapt the weights assigned because some growth methods may consistently perform better than others in certain areas. We want to be able to use this information to automatically tune the weight each method has of being selected.

VIII. CONCLUSIONS

Using obstacle hints for directions to grow a tree for path planning can be beneficial, especially when exploring difficult areas. This is clear from the environments that require difficult movements of the robot or a sliding motion. There are also many other ways of growing a tree that should be considered. Using the Obstacle-Based RRT described here along with a strong node generation method, that can find difficult areas of C-space, could result in a path planning technique that solves motion planning problems quickly and efficiently.

REFERENCES

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics: The Algorithmic Perspective*, pages 155–168, Natick, MA, 1998. A.K. Peters. Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Trans. Robot. Automat.*, 16(4):442–447, August 2000. Preliminary version appeared in ICRA 1998, pp. 630–637.

[3] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, 1991.

[4] O. B. Bayazit, G. Song, and N. M. Amato. Enhancing randomized motion planners: Exploring with haptic hints. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 529–536, 2000.

[5] J. Berg and M. Overmars. Using workspace information as a guid to non-uniform sampling in probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 453–460, 2004.

[6] P. Bessiere, J. M. Ahuactzin, E. G. Talbi, and E. Mazer. The Ariadne's clew algorithm: Global planning with local methods. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, volume 2, pages 1373–1380, 1993.

[7] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1018–1023, 1999.

[8] H. Chang and T. Y. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1012–1019, 1995.

[9] D. Hsu, T. Jiang, J. Reif, and Z. Sun. Bridge test for sampling narrow passages with proabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4420–4426, 2003.

[10] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 141–153, 1998.

[11] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2719–2726, 1997.

[12] L. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2138–2145, 1994.

[13] 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. Automat.*, 12(4):566–580, August 1996.

[14] J. J. Kuffner and S. M. LaValle. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 995–1001, 2000.

[15] S. M. LaValle and J. J. Kuffner. Rapidly-Exploring Random Trees: Progress and Prospects. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages SA45–SA59, 2000.

[16] J.-M. Lien, O. B. Bayazit, R.-T. Sowell, S. Rodriguez, and N. M. Amato. Shepherding behaviors. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4159–4164, April 2004.

[17] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 316–376, Utrecht/Zeist, The Netherlands, July 2004.

[18] M. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 19–37, 1994.

[19] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Trans. Robot. Automat.*, 2005.

[20] S. Redon and M. C. Lin. Practical local planning in the contact space. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, April 2005.

[21] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 421–427, San Juan, Puerto Rico, October 1979.

[22] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1024–1031, 1999.

[23] A. Yershova, L. Jaillet, T. Simeon, and S. M. Lavalle. C-space subdivision and integration in feature-sensitive motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3867–3872, April 2005.