FCRAR2021-1010

Path Planning Algorithms: An Evaluation of Five Rapidly Exploring Random Tree Methods

Jacqueline Jermyn, Rodney Roberts

FAMU-FSU College of Engineering
2525 Pottsdamer Street
Tallahassee, Florida 32310

jmj15c@my.fsu.edu, rroberts@eng.famu.fsu.edu

ABSTRACT

Path planning is one element of mobile robot navigation. It helps a robot arrive at its destination safely by defining its route from the start to the end locations. The Rapidly Exploring Random Tree (RRT) algorithm is a frequently employed type of path planning technique because it is uncomplicated, even though it does not come up with the optimal route. For this reason, variations of RRT algorithms have been developed to improve its effectiveness. This paper evaluates the performance of five variations of RRT algorithms. These include the RRT, Rapidly Exploring Random Tree Star (RRT*), Informed RRT*, Linear Quadratic Regulator Rapidly Exploring Random Tree Star (LQR RRT*), and the Batch Informed Tree Star (BIT*).

Keywords

Rapidly Exploring Random Tree (RRT), Rapidly Exploring Random Tree Star (RRT*), Informed RRT*, Linear Quadratic Regulator Rapidly Exploring Random Tree Star (LQR RRT*), Batch Informed Tree Star (BIT*), Path Planning, Mobile Robot Navigation

1. INTRODUCTION

Path planning algorithms plan a route through a robot's environment to its destination. Its environs contain free space and space that is occupied by objects. Sampling-based algorithms use object collision detection functions to plan an obstacle-free path [12]. They build their own graphs to model the relationships among entities in their milieu because they do not use pre-built graphs [17]. Sampling-based methods are probabilistically complete as they will find a path to a robot's destination provided that such a path exists, if the number of iterations executed is sufficient to find this path [4], [5], [8], [11], [15].

The two well established types of sampling-based path planning techniques are probabilistic roadmap (PRM) and rapidly exploring random tree (RRT) methods [16], [17]. The PRM technique is a multi-query method because it begins the search for a path by randomly selecting points from the free space as nodes to construct a roadmap [12]. It forms this graph by attaching each node to all the other nodes that are within a radius specified by the programmer. After the roadmap has been developed, a graph search technique, such as the A* method, specifies a path from the start to goal locations [16].

The RRT algorithm, on the other hand, iteratively grows a search tree from its initial position by adding nodes sampled from the free space until it reaches its end point. This method also outputs the path as soon as the search tree reaches the goal location without using a graph search method. Instead, this algorithm works backwards from the goal location by finding the parent node for each node in the path until it reaches the starting point [12]. The paths that are outputted by the RRT algorithm are not asymptotically optimal because new points are attached to nearest node in the tree without additional optimization steps [17].

Asymptotically optimal RRT methods improve the path as the number of iterations increases [8]. They include the RRT*, Informed RRT*, LQR RRT*, and BIT* methods. The RRT* method enhances the RRT method by shortening the path between two points [17]. It continues for a pre-selected number of iterations, calculates the total distance to various nodes, and rewires the tree to optimize the path [5], [17].

The Informed RRT* algorithm is a variation of the RRT* technique that samples points from an elliptical search area [4], [6]. For the Informed RRT* algorithm, the foci of its search ellipse are the start and goal positions. The minimum distance between these two locations is the line segment between them. Selecting points from this area allows the Informed RRT* method to concentrate on the area that is proximate to the initial and final positions. It shortens the path length by preventing a robot from traveling to points that are far away from the start and goal locations [4].

The Linear Quadratic Regulator Rapidly Exploring Random Tree Star (LQR RRT*) method is a variant of the RRT* technique. It linearizes the control system about points that are sampled from the environ. Linearization is performed so that Linear Quadratic Regulation (LQR), which is a control technique for linear systems, can be applied to optimize the path by choosing the nearest node to extend the search tree [1]. Linear control systems are additive and homogeneous.

The Batch Informed Tree Star (BIT*) method combines the Informed RRT* technique with a graph search method [5]. It samples multiple batches of points, and the number of points per batch is preselected by the programmer. These points are used to

construct an explicit search tree and an implicit random geometric graph (RGG). This graph is implicit because edges in the graph are specified by a transition function [3], [5]. Processing points in batches allows the distribution of the sampled points to depend on a cost function. In addition, selecting multiple batches of points causes the path to asymptotically reach the shortest possible path as the number of iterations increases. This method uses a graph search technique that is similar to the Lifelong Planning A* (LPA*) method to search a sequence of RGGs [5]. The LPA* graph search technique facilitates finding a path because it incrementally integrates samples from new batches of points into the search [5], [10].

The relationship among these sampling-based path planning techniques is shown in Figure 1.

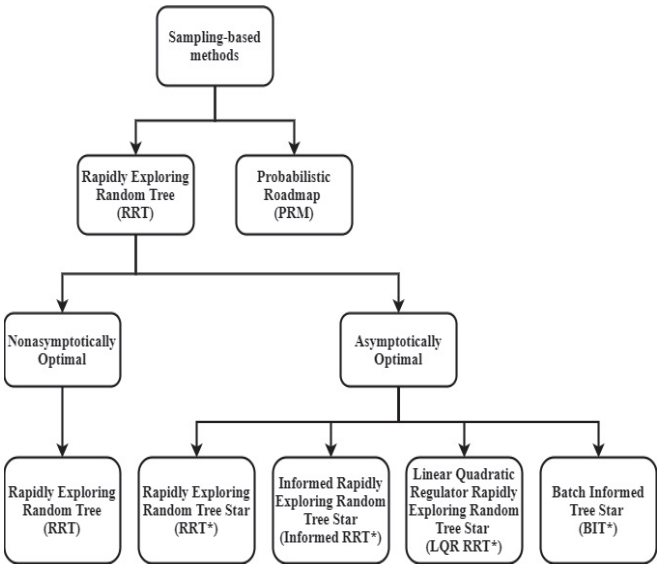

Figure 1: Sampling-based Path Planning Algorithm Taxonomy

Previous experiments that were conducted by the authors compared the effectiveness of three rapidly exploring random tree algorithms to find a route for two simulated environments that contained fixed circular obstacles [6]. The techniques that were studied were the RRT, RRT*, and Informed RRT* algorithms. These search areas were rectangular, and they were the same size for both sets of the experiments. The vertices of these search areas were (-5, -2), (-5, 15), (15, 15), and (15, -2). For both sets of these experiments, the start position for all path planning techniques was (0,0), and the finish point was placed at (10,10) [6].

The previous experiments that were conducted used three RRT techniques to find a path. Each set of experiments was run until a path was found 30 times. The maximum number of searches for each algorithm was 200 [6]. The RRT and RRT* methods found paths 100% of the time for both sets of obstacles under these experimental conditions. The Informed RRT* technique found a path 41.76% of the time for obstacle set 1 and 94% of the time for obstacle set 2. The Informed RRT* technique also outputted the shortest paths for both obstacle sets. The results for these experiments also showed that the arrangement of objects affects the length of a route and the amount of time it takes to find a path, irrespective of the variation of path planning method [6].

This paper presents the findings for follow-on studies to previous experiments conducted by the authors. They are different because more types of RRT algorithms are evaluated under more rigorous experimental conditions. In this paper, the five types of methods that are studied include the RRT, RRT*, Informed RRT*, LQR RRT*, and the BIT* methods. Even though experiments had previously been conducted using the RRT, RRT*, and the Informed RRT* methods, they were re-accomplished using new experimental conditions so that comparisons could be made for all five algorithms. This year's experimental setup is more detailed. The number of searches has been increased from 200 to 400 to give each algorithm sufficient opportunities to find a route if one exists. In addition, the search area was increased so there is more room for the robot to travel around the environment to find a path. Thirty randomly generated static obstacle sets were used instead of only two obstacle sets to meet the central limit theorem. In addition, each path planning technique is applied 30 times to each of the 30 obstacle sets. This was accomplished because the previous experiments showed that there is variation in path length when the same RRT path planning algorithm is repeatedly simulated on the same obstacle set [6].

The remainder of the paper is structured in the following manner. Section 2 describes the five types of RRT algorithms. Section 3 discusses the experimental set up. Section 4 reviews the results of the experiments. Section 5 concludes with the highpoints of the simulations, and lastly, Section 6 explores related future work.

2. DISCUSSION OF ALGORITHMS

2.1 Rapidly Exploring Random Tree (RRT)

The RRT algorithm begins by initializing the variables. They include the search area boundaries, obstacle list, start location, goal locations, and search tree. The start position is added to the search tree. An iteration starts by selecting a random point in the environment. Next, this algorithm determines the nearest point in the search tree to the randomly selected point by iterating over the list of the nodes in the tree. The distance between each node in the tree to the randomly selected point is calculated as:

$$d_{nearest,i} = \sqrt{\left(node_{i,x} - rnd_x\right)^2 + \left(node_{i,y} - rnd_y\right)^2} \quad (1)$$

where $d_{nearest,i}$ represents the distance between the random point and the node with index i in the node list, $node_{i,x}$ is the x coordinate of ith node in this list, $node_{i,y}$ is the y coordinate of the ith node in this list, rnd_x is the x coordinate of the random point, and rnd_y is the y coordinate of the random point. The nearest node in the tree has the minimum $d_{nearest,i}$ value.

The steer function finds a potential new point in the environment to connect to the nearest node [8]. For points where the random node is within the maximum edge expansion distance that is selected by the programmer, the random point is used as the new node. If the distance between the nearest node and random node is greater than the maximum expansion distance, the position of the potential new node is calculated [18]. To do so, the angle, θ, between the nearest node and the random point is calculated using the following:

$$dx = rnd_x - nearest_x \quad (2)$$

$$dy = rnd_y - nearest_y \quad (3)$$

$$\theta = \text{atan2}(dx, dy) \quad (4)$$

where dx is the displacement along the x axis from the x coordinate of the nearest node, $nearest_x$, to the x coordinate of the random node rnd_x. In addition, the displacement, dy, is the change in position along the y axis from the y coordinate of the nearest node, $nearest_y$, to the y coordinate of the random point, rnd_y. The x and y coordinates of the new node are calculated as

$$new_x = nearest_x + d_{exp} \cos\theta \quad (5)$$

$$new_y = nearest_y + d_{exp} \sin\theta \quad (6)$$

where new_x is the x-coordinate of the potential new node, new_y is the y-coordinate of the potential new node, and d_{exp} is the maximum edge expansion distance chosen by the user.

The line segment between the nearest point in the tree and the new node is checked for collision with an object. If this edge results in a collision, the new node is not added to the search tree, and the iteration ends. If the edge between the nearest node and the new node is obstacle free, these two points are added to the search tree, and an edge is extended between them. This process repeats until the goal position has been reached. When this happens, the goal position is added to the search tree. The RRT algorithm ends by outputting the search tree and the path from start to goal. Figure 2 shows the activity diagram for the RRT algorithm.

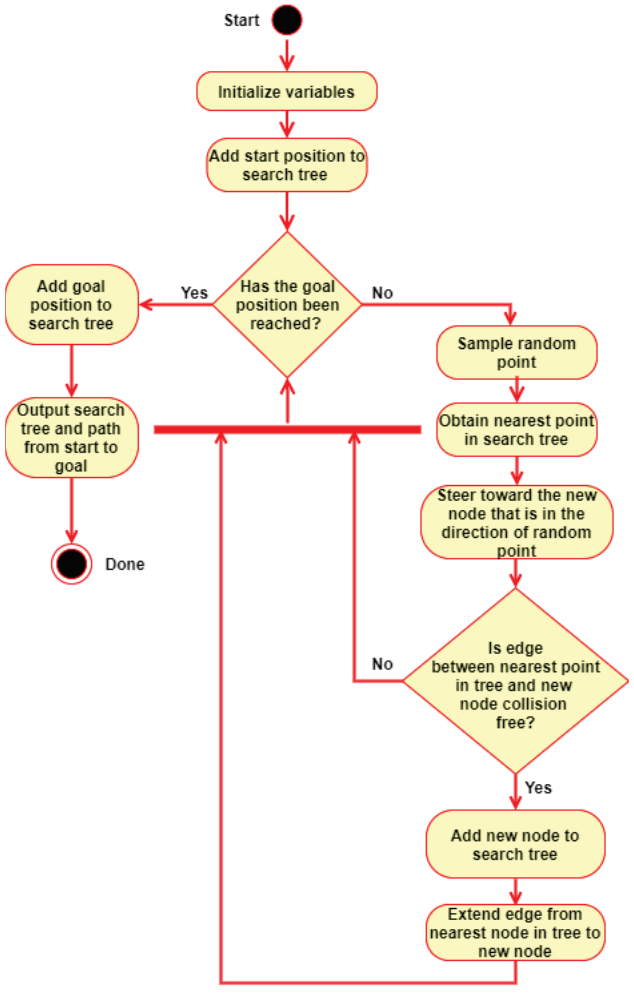

Figure 2: RRT Activity Diagram

2.2 Rapidly Exploring Random Tree Star (RRT*)

The RRT* method starts by initializing variables and adds the initial position to the search tree. It then selects a random point in the environment and steers toward it. If traveling along this edge causes an accident, this point will be discarded. This location becomes a new node for the tree, if the edge between it and the nearest point in the tree does not cause a collision with an object. It subsequently compares the costs of a group of nodes that are within a radius of the new node [8]. The equation for the radius, r, is given as

$$r = \gamma \sqrt{\frac{\log n}{n}} \quad (7)$$

where γ is the connect circle distance constant that is selected by the programmer, and n is the number of nodes in the search tree [15]. The cost is the distance each node has traveled in relation to its parent node. The near node is the node with the lowest cost in the search radius, and it becomes the parent node for the new node.

The rewiring function checks nodes in the near neighborhood around the new node to determine if making the new node their parent node decreases the cost. If this is the case, they are rewired through the new node [14].

When the maximum number of iterations has been reached, this algorithm outputs the search tree and path from start to goal. The RRT* activity diagram is shown in Figure 3.

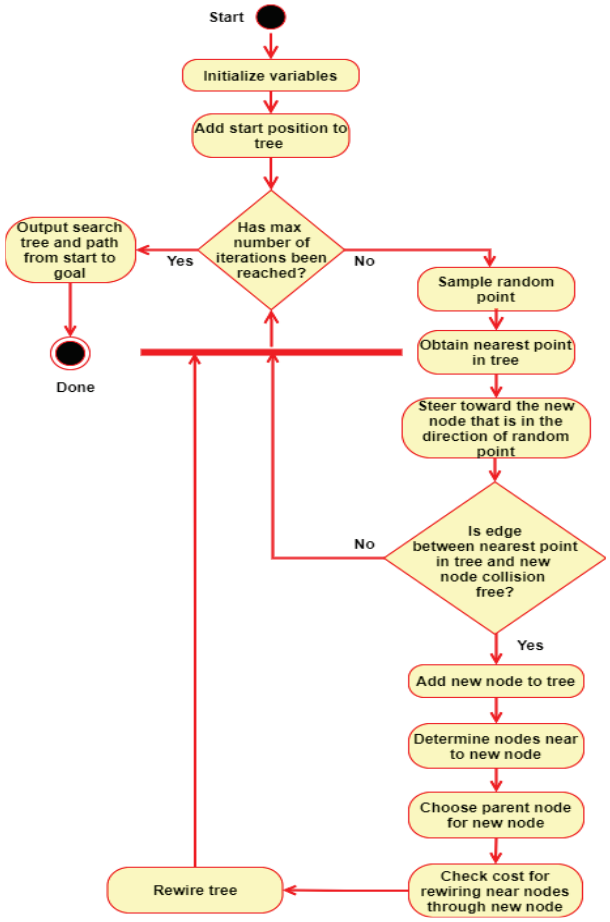

Figure 3: RRT* Activity Diagram

2.3 Informed RRT*

The Informed RRT* algorithm first initializes the variables. The start position is then added to the search tree. Next, it chooses a point in the elliptical subset of the environment close to the start and goal locations. The equation for the eccentricity of the search ellipse is given as

$$eccentricity = \frac{c_{min}}{c_{best}} \tag{8}$$

where c_{min} is the distance between the foci, and c_{best} is the cost of the current path. The eccentricity of an ellipse ranges from zero to one. The eccentricity increases as lower and lower cost paths are found over time [4].

This algorithm then selects the nearest node in the search tree to the point chosen from the elliptical search area. It determines a new node that is in the direction of the point from the elliptical search area.

If the edge between the nearest point in the tree and the new node causes a collision, it is not used, and a new iteration starts. If this edge is collision-free, it connects the new node to the tree. Next, this method compares the costs of the nodes that are within a radius of the new node. The node with the smallest cost in the search radius is set as the new node's parent node. The rewiring function examines all neighboring nodes around the new node to determine if they decrease the cost. If this is the case, this function changes their parent node to the new node. When this method reaches the maximum number of iterations, the search tree and path from start to goal are outputted [4]. The activity diagram is shown in Figure 4.

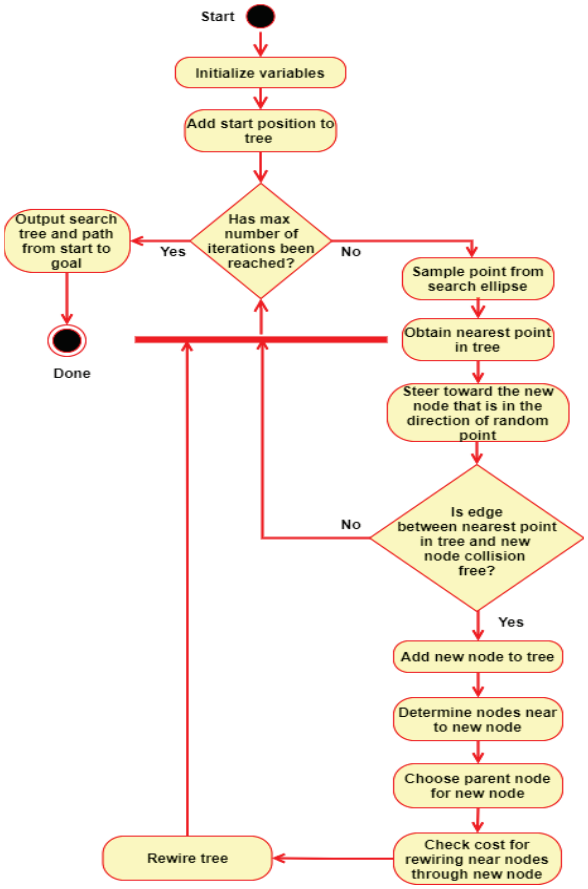

Figure 4: Informed RRT* Activity Diagram

2.4 Linear Quadratic Regulator Rapidly Exploring Random Tree Star (LQR RRT*)

The first steps for the LQR RRT* method involve initializing variables and using the starting location as the root node of the tree. This algorithm subsequently chooses a random point in the environment [15]. It then linearizes the system about the random point, rnd, and the action u = 0. The gain and cost matrices are calculated as

$$[K(rnd), S(rnd)] = LQR(A, B, Q, R) \tag{9}$$

where K(rnd) is the gain matrix, S(rnd) is the cost matrix, A is the state matrix, and B is the input matrix. In addition, Q and R are positive definite matrices [20]. The index of the nearest node, $x_{nearest}$, in the tree to the random point is obtained using the following:

$$x_{nearest} = \underset{v \in V}{\operatorname{argmin}} (v - rnd)^T S (v - rnd) \tag{10}$$

where v is a node in the node list, V, and rnd is the random point in the environment. The LQR RRT* method then determines the new node in the direction of the random node.

If the line segment between the nearest node and the new node hits an obstacle, it is discarded. If this line segment does not cause a collision, the new node is added to the tree. It subsequently compares the costs of a group of nodes that are within a radius of the new node, and equation for radius is the same as the one for the RRT* radius [8], [15]. The node with the lowest cost within this radius is set as the parent node for the new node.

Next, the costs for rewiring the near nodes through the new node are checked. If a path that connects a near node to the new node is less than the cost of its current parent node, the new node is set as the parent node for this node. The algorithm ends when it arrives at the preset number of iterations. It then outputs the search tree and final path [15]. The LQR RRT* activity diagram is provided in Figure 5.

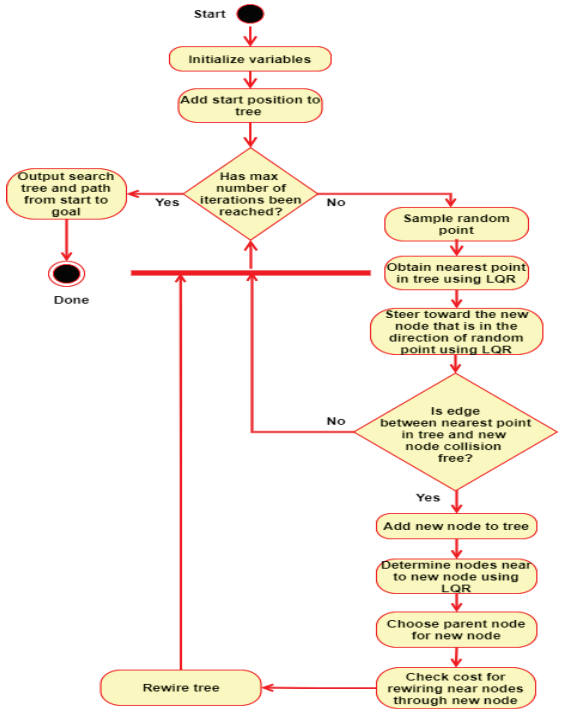

Figure 5: LQR RRT* Activity Diagram

2.5 Batch Informed Tree Star (BIT*)

The steps for the BIT* method include algorithm setup, batch creation, edge selection, and edge processing. The first step is algorithm setup, and it involves defining variables and adding the start position to the tree.

The batch creation step occurs when the edge queue and the vertex expansion queue are empty. This step prunes the tree by removing locations that cannot decrease the path length. It also samples random points in the environment to add to the RGG. Nodes in the tree are called vertices, and they are labeled to consider only edges that connect them to new locations. In addition, the size of the radius, r, of the disk for the RGG is computed using the following:

$$r = 2\eta\left(1+\frac{1}{n}\right)^{\frac{1}{n}}\left(\frac{\lambda\left(X_{\hat{f}}\right)}{\zeta_n}\right)^{\frac{1}{n}}\left(\frac{\log(q)}{q}\right)^{\frac{1}{n}} \quad (11)$$

where q is the number of points in the RGG, and η is a constant that is greater than or equal to one [5], [9]. The number of dimensions of the environment is n. For a two-dimensional environment, the value of n is equal to two.

The Lebesgue measure of the set of samples, $X_{\hat{f}}$, is $\lambda\left(X_{\hat{f}}\right)$. In addition, the Lebesgue measure of a unit ball that has n dimensions is ζ_n [5], [9]. In other words, the Lebesgue measure is a procedure that gives a number to the subset of a set of samples that are in n-dimensional Euclidean spaces, which have dimensions that are positive integers. When the environment is two-dimensional, the ζ_n corresponds to the area of a unit circle.

The edge selection step processes the edge queue order of increasing estimated path cost for each path that passes through a particular edge in the current tree. It also removes the best edge in the queue for analysis by the edge processing step [5].

The actual cost of each edge is calculated by the edge processing step. It first checks if the best edge improves the path. If it does not improve the path, the edge and vertex queues are cleared and a new batch starts. If the best edge improves the current path, it is checked to ensure that it does not cause a collision with an obstacle. If it causes a collision, a new iteration starts. If this edge is obstacle free, it is checked to ensure it improves the cost-to-come of its target node. If it does not, a new iteration is started. If it improves the cost-to-come, the vertex is checked to determine if it is already in the search tree. If it is in the tree, this vertex is rewired. If it is not in the tree, the target vertex is expanded. The next steps are to add the edge to the tree and to prune the edge queue [5].

The BIT* technique ends when the maximum number of iterations has been achieved. It outputs the search tree and the path to the destination. Figure 6 shows the BIT* activity diagram.

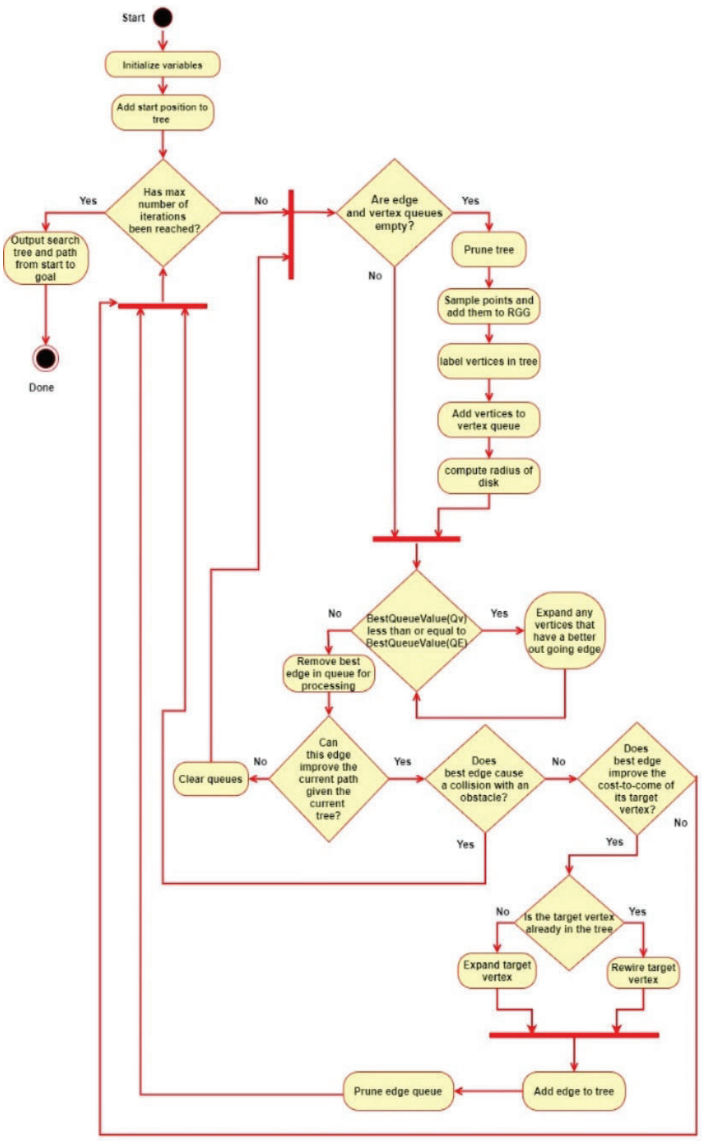

Figure 6: BIT* Activity Diagram

3. EXPERIMENTAL SETUP

Experiments in a two-dimensional environment were conducted using the PythonRobotics toolbox for five RRT algorithms [18]. These include the RRT, RRT*, Informed RRT*, LQR RRT*, and BIT* algorithms.

The rationale for conducting computer simulations, instead of performing real-world experiments with a robot, is to eliminate robot motion execution errors, robot localization errors, and obstacle localization errors. As a result, any differences in the experimental results are directly attributed to the type of path planning algorithm that is implemented.

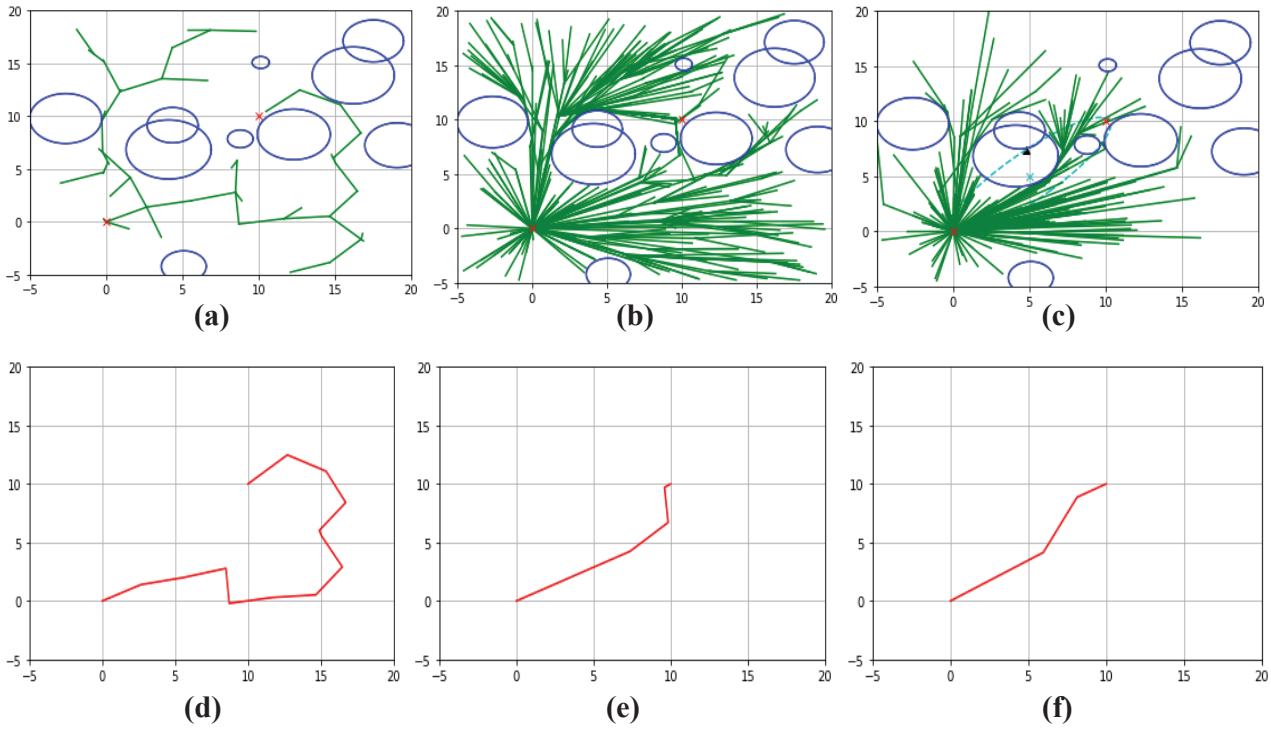

Figure 7: Simulations for Obstacle Set 1 Trial 1 for the RRT, RRT*, and Informed RRT* Techniques. Figures (a) and (d) are the search tree and final path for the RRT algorithm. Figures (b) and (e) are the search tree and final path for the RRT* algorithm. Figures (c) and (f) are the search tree and the final path for the Informed RRT* algorithm.

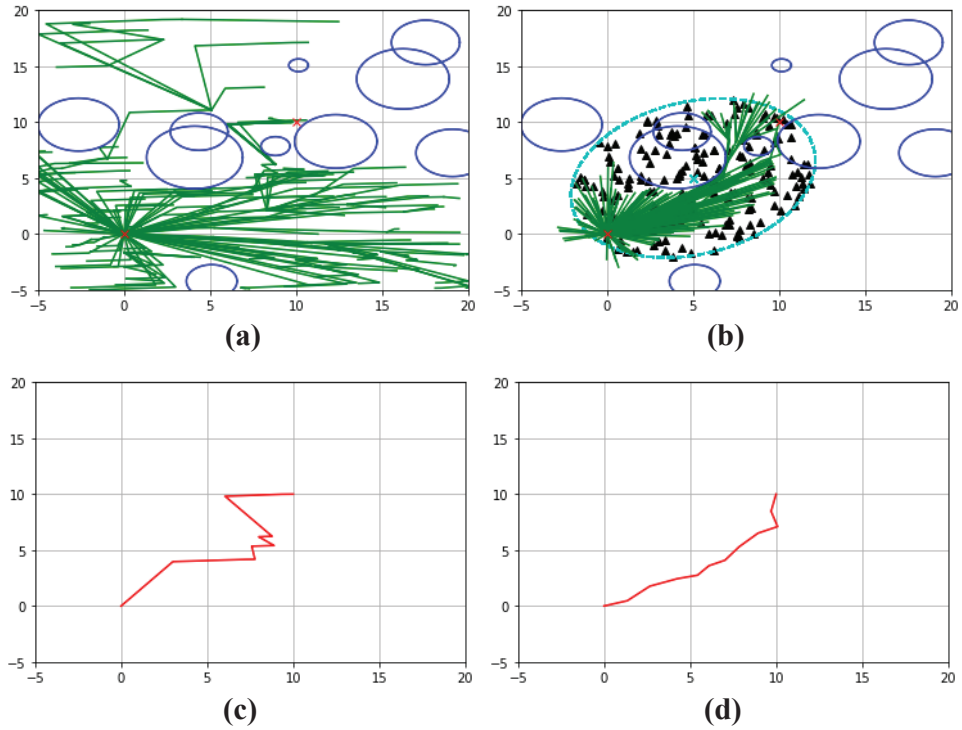

Figure 8: Simulations for Obstacle Set 1 Trial 1 for the LQR RRT* and BIT* Techniques. Figures (a) and (c) are the search tree and final path for the LQR RRT* algorithm. Figures (b) and (d) are the search tree and final path for the BIT* algorithm.

The simulation environment is a rectangular one, and its vertices are (-5, -5), (-5, 20), (20, 20), and (20, -5). The coordinate of the initial position for all paths is (0,0), the goal position is located at (10,10).

Metric maps were implemented because they represent locations of objects in the environment using coordinates, and this simplifies the detection of collisions with obstacles [19]. Thirty randomly generated obstacle sets of 10 circular obstacles are employed for these experiments. When these obstacle sets were initially generated, the x coordinate, y coordinate, and radius of each circle were obtained by sampling uniform distributions. The probability distribution function for a uniform distribution is given as

$$p(x) = \frac{1}{b-a} \tag{12}$$

where b is the upper bound, and a is the lower bound of the interval. The value of p(x) is 0 outside of the interval. For the x and y coordinates, the interval for the distribution is [-5, 20]. For the radius, the interval for the distribution is [0.5, 3].

The obstacle sets were also visually inspected to ensure that an obstacle was not placed on top of the start or goal locations because the path planning methods would be unable to find a path.

For each path planning algorithm, there are 30 trials run for each of the 30 obstacle sets resulting in a total of 900 trials. During each trial, the maximum number of iterations is set to 400 so as to provide ample opportunities for the algorithm to find a path. In addition, when the number of samples collected increases, its probability distribution function approximates a Gaussian distribution. Furthermore, 30 trial runs and 30 obstacle sets were chosen so that the central limit theorem could be applied. The search trees and paths for Obstacle Set 1, Trial 1, for the RRT, RRT*, and Informed RRT* are provided in Figure 7. The search trees and paths for Obstacle Set 1, Trial 1, for the LQR RRT* and the BIT* are provided in Figure 8.

The runtime, number of iterations, total number of nodes in the search trees, total number of nodes in the path, and path length are measured for each trial. The runtime is the time it takes for the planning function for each algorithm to execute. It does not include time for initializing variables and for displaying the figures. The number of iterations is the number of times that a random point is sampled in the search area. The number of nodes in the search tree is the number of points that have been added to a search tree. The number of nodes in the path is the number of points in the final path. The path length is the sum of the Euclidian distances between each consecutive pair of points in the final path. The algorithm with the shortest path length is considered as the most effective. At the end of each trial, the runtime, number of iterations, total number of nodes in the search tree, total number of nodes in the path, and path length are appended to lists to facilitate analyzing the data.

4. RESULTS

The sample statistics are calculated for each independent variable list. They are the mean, median, minimum, maximum, mode, and standard deviation. The sample statistics for each algorithm are provided in Tables 1 through 5 in the Appendix. The summary of the experimental results for all five methods is provided in Table 6, also located in the Appendix.

All the path planning techniques successfully found a path. For the RRT*, Informed RRT*, LQR RRT*, and the BIT* algorithms, the number of iterations was set to 400 to increase the probability for them to find a path. For the RRT algorithm, it terminates as soon as it finds a path, and it found a path after an average of 19.623 iterations.

For the RRT, the RRT*, and the Informed RRT*, the mean runtimes were 0.004 s, 0.662 s, and 0.488 s respectively. The average runtimes for the for the LQR RRT* and for the BIT* are 0.510 s and 0.501 s respectively. In terms of runtime, the RRT method has the best performance because it takes the least amount of time to generate a path to the destination.

The average number of nodes in the tree for the RRT, the RRT*, and the Informed RRT* are 14.322, 337.834, and 290.246 respectively. The mean number of nodes in the search trees for the LQR RRT* and for the BIT* are 334.412 and 398.239 respectively. In terms of the number of nodes in the tree, the algorithm with the best performance has the fewest number of nodes because they require less computer memory to store [2]. As a result, the RRT method is the most efficient in terms of the number of nodes in the search tree.

The number of nodes in the final path for the RRT, RRT*, and Informed RRT* techniques are 8.114, 4.113, and 4.512 respectively. The number of nodes in the final paths for the LQR RRT* and for the BIT* algorithms are 9.581 and 12.942 respectively. When a robot follows a polyline path, it rotates in place at each node and executes a pure forward translation along each edge [13]. Paths with fewer nodes require a robot to make fewer motion executions; consequently, they are more efficient [2], [13]. As a result, the RRT* method is the most efficient with respect to the number of nodes in the path.

The mean path length for the RRT, RRT*, and the Informed RRT* are 19.606, 15.107, and 14.675 respectively. The mean path lengths for the LQR RRT* and for the BIT* are 20.366 and 15.956 respectively. Robots with shorter path lengths are more efficient because they travel shorter distances to get to their destinations [2]. Because of this reason, the Informed RRT* method is the most effective in terms of path length.

The most important metric for evaluating these algorithms is the path length because it determines how rapidly a robot will reach its destination without colliding with obstacles. The Informed RRT* has the best overall performance, for the experimental conditions described in this paper, because it outputs the shortest paths.

5. CONCLUSIONS

Sampling-based path planning algorithms detect obstacles in their environment to generate a collision-free route from their start point to their destinations. They are divided into PRM and RRT techniques. This paper highlights the results for the follow-on experiments that had been accomplished by the authors. This study involves evaluating five types of RRT algorithms using more detailed experimental conditions. They include the RRT, RRT*, Informed RRT*, LQR RRT*, and the BIT* methods. Two-dimensional simulations were performed using the PythonRobotics toolbox. The runtime, number of iterations, number of nodes in the search tree, number of nodes in the path, and the path lengths were measured.

All methods successfully found a path because increasing the maximum number of searches to 400 allowed for sufficient time for each technique to find a path. By contrast, when only 200 searches were used in prior experiments, the Informed RRT* technique did not find a path every time [6]. The RRT method found a path after an average of 19.623 iterations, it had the shortest average runtime of 0.004 s, and it had the lowest average number of nodes in its search trees of 14.322 nodes. The RRT* algorithm had the lowest average number of nodes in the path of 4.113. The Informed RRT* method yielded the shortest mean path length of 14.675 rendering this method the one with the best overall performance. For the previous experiments, the Informed RRT* technique also yielded the shortest path lengths [6].

6. FUTURE WORKS

Future work will involve evaluating the effectiveness of Bidirectional RRT path planning algorithms [7], [12]. These types of algorithms alternate between growing the search trees from the start position and from the end position. A path is found when the two search trees connect with each other. The Bidirectional RRT path planning algorithms are more effective than other RRT algorithms at escaping one-way doors [12].

These experiments could be conducted by creating two search trees. The first tree will be called the initial tree because it uses the initial position as its root node. The second tree is the goal tree because it begins at the goal position. A random point will be generated, checked for collisions, and added to the initial tree as a leaf node. The goal tree will use the same node to extend itself. A second random point will then be selected, checked for collisions, and added to the goal tree as another leaf node. The initial tree will use the leaf node from the goal tree to extend itself. If one search tree successfully connects to the leaf node from the other tree, a path is found from the start to the goal position.

7. ACKNOWLEDGMENTS

The authors would like to thank the FAMU-FSU College of Engineering's Electrical and Computer Engineering Department, the Engineering Dean's Fellowship Program, and the Department of Defense's Science, Mathematics, And Research for Transformation Scholarship Program for their encouragement and for their support.

8. REFERENCES

[1] S. Boyd. (2009, March). Linear Quadratic Regulator: Discrete-time Finite Horizon. [Online]. Available: https://stanford.edu/class/ee363/lectures/dlqr.pdf

[2] N. Correll. (2020, Mar. 6). *Introduction to Autonomous Robots.* (2nd ed.). [Online]. Available: https://github.com/correll/Introduction-to-Autonomous-Robots/releases

[3] J. D. Gammell, "Informed Anytime Search for Continuous Planning Problems," Ph.D. dissertation, Aerospace Sci. and Eng., Univ. Toronto, Toronto, Ont., 2017.

[4] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. (2014, Sept.). Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic. Presented at *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems.* [Online] Available: https://ieeexplore.ieee.org/document/6942976

[5] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. (2015, May). Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs. Presented at *2015 IEEE International Conference on Robotics and Automation (ICRA).* [Online]. Available: https://ieeexplore.ieee.org/document/7139620

[6] J. Jermyn and R. Roberts. (2020, May). A Review of Rapidly Exploring Random Tree Path Planning Algorithms. Presented at *33rd Florida Conference on Recent Advances in Robotics.* [Online]. Available: https://fcrar2020.fit.edu/proceedings.pdf

[7] M. Jordan and A. Perez. "Optimal Bidirectional Rapidly-Exploring Random Trees," Comp. Sci. Art. Int. Lab., MIT. MA, Cambridge, Tech. Report. MIT-CSAIL-TR-2013-021, Aug. 15, 2013.

[8] S. Karaman and E. Frazzoli. (2010). Incremental Sampling-based Algorithms for Optimal Motion Planning. Presented at *2010 Robotics: Science and Systems Conference.* [Online]. Available: http://roboticsproceedings.org/rss06/p34.pdf

[9] S. Karaman and E. Frazzoli. (2011, Jun.). Sampling-based Algorithms for Optimal Motion Planning. *The International Journal of Robotics Research.* [Online]. 30 (7), 766-791. Available: https://journals.sagepub.com/doi/abs/10.1177/0278364911406761#articleCitationDownloadContainer

[10] S. Koenig, M. Likhachev, and D. Furcy. (2004, Oct.). Lifelong Planning A*. *Artificial Intelligence*. [Online]. 155 (1-2), pp. 93-146. Available: https://www.sciencedirect.com/science/article/pii/S000437020300225X

[11] S. M. LaValle, "Continuous Discrete," in *Planning Algorithms,* 1st ed. New York, Cambridge Univ. Press, 2006, ch. 2, sec. 2, pp. 79-80.

[12] S. M. LaValle. (2011). Motion Planning: The Essentials. *IEEE Robotics & Automation Magazine.* [Online]. *18(1),* pp. 79-89. Available: http://msl.cs.illinois.edu/~lavalle/papers/Lav11b.pdf

[13] W. S. Newman. (2018). *A Systematic Approach to Learning Robot Programming with ROS.* (1st ed.). [Print].

[14] I. Noreen, A. Khan, and Z. Habib. (2016, Oct). A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms. *International Journal of Computer Science and Network Security.* [Online]. 16 (10), pp. 20-27. Available: http://paper.ijcsns.org/07_book/201610/20161004.pdf

[15] A. Perez et al. (2012, May) LQR-RRT*: Optimal Sampling-Based Motion Planning with Automatically Derived Extension Heuristics. Presented at *2012 IEEE International Conference on Robotics and Automation (ICRA).* [Online]. Available: https://ieeexplore.ieee.org/document/6225177

[16] Robotic Path Planning: PRM & PRM*: 2019. https://theclassytim.medium.com/robotic-path-planning-prm-prm-b4c64b1f5acb. Accessed: 2021-02-16.

[17] Robotic Path Planning: RRT and RRT*: 2019. https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378. Accessed: 2021-02-16.

[18] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques. (2018, Sep.). PythonRobotics: a Python code collection of robotics algorithms. ArXiv, abs/1808.10703.
[19] S. Thrun. (1998). Learning Metric-topological Maps for Indoor Mobile Robot Navigation. *Artificial Intelligence.* [Online]. 99*(1)*. pp. 21-71. Available: https://www.sciencedirect.com/science/article/pii/S0004370297000787
[20] M. Triantafyllou. (2004). Linear Quadratic Regulator. *MIT OpenCourseWare.* [Online]. Available https://ocw.mit.edu/courses/mechanical-engineering/2-154-maneuvering-and-control-of-surface-and-underwater-vehicles-13-49-fall-2004/lecture-notes/lec19.pdf

9. APPENDIX

The results for the RRT, RRT*, Informed RRT*, LQR RRT*, and BIT* are provided in Tables 1, 2, 3, 4, and 5 respectively. The summary of experimental results is provided in Table 6.

Table 1: RRT Results

Quantity	Mean	Median	Min	Max	Mode	Standard Deviation
Runtime (s)	0.004	0.002	0.000	0.059	0.001	0.006
Number of Iterations	19.623	14.000	4.000	242.000	7.000	20.506
Number of Nodes in Tree	14.322	11.000	5.000	163.000	9.000	12.612
Number of Nodes in Path	8.114	8.000	6.000	19.000	7.000	1.651
Path Length	19.606	18.515	14.146	47.657	None	4.317

Table 2: RRT* Results

Quantity	Mean	Median	Min	Max	Mode	Standard Deviation
Runtime (s)	0.662	0.677	0.004	0.990	None	0.172
Number of Iterations	400	400	400	400	400	0.000
Number of Nodes in Tree	337.834	338.500	286.000	378.000	344.000, 332.000	15.259
Number of Nodes in Path	4.113	4.000	3.000	6.000	4.000	0.614
Path Length	15.107	14.748	14.142	18.977	None	0.978

Table 3: Informed RRT* Results

Quantity	Mean	Median	Min	Max	Mode	Standard Deviation
Runtime (s)	0.488	0.502	0.001	0.999	0.127	0.287
Number of Iterations	400	400	400	400	400	0.000
Number of Nodes in Tree	290.246	278.000	172.000	401.000	401.000	66.932
Number of Nodes in Path	4.512	4.000	3.000	7.000	4.000	0.676
Path Length	14.675	14.227	14.142	19.016	none	0.791

Table 4: LQR RRT* Results

Quantity	Mean	Median	Min	Max	Mode	Standard Deviation
Runtime (s)	0.510	0.518	0.001	0.999	0.745	0.288
Number of Iterations	400	400	400	400	400	0.000
Number of Nodes in Tree	334.412	335.500	247.000	380.000	335.000	17.259
Number of Nodes in Path	9.581	9.000	6.000	18.000	6.000	2.786
Path Length	20.366	20.323	18.586	30.692	20.747	1.114

Table 5: BIT* Results

Quantity	Mean	Median	Min	Max	Mode	Standard Deviation
Runtime (s)	0.501	0.490	0.002	0.998	0.181	0.299
Number of Iterations	400	400	400	400	400	0.000
Number of Nodes in Tree	398.239	399.000	383.000	401.00	399.000	1.606
Number of Nodes in Path	12.942	13.000	11.000	19.000	13.000	1.187
Path Length	15.956	15.569	14.194	22.859	none	1.340

Table 6: Summary of Results

Algorithm	Runtime (s)	# Iterations	# Nodes in Tree	# Nodes in Path	Path Length
RRT	0.004	19.623	14.322	8.114	19.606
RRT*	0.662	400	337.834	4.113	15.107
Informed RRT*	0.488	400	290.246	4.512	14.675
LQR RRT*	0.510	400	334.412	9.581	20.366
BIT*	0.501	400	398.239	12.942	15.956