Bi-RRT* Path Planning Algorithm for Large Scale Apple Picking Robot

Dillon Miller
University of Maryland
College Park, MD, USA
Dmille19@umd.edu

Caleb Myers
University of Maryland
College Park, MD, USA
Cmyers17@umd.edu

Hamsaavarthan Ravichandar
University of Maryland
College Park, MD, USA
rhamsaa@umd.edu

***Abstract*—Agriculture within the United States is slowly approaching autonomy, and path planning for agricultural robotics is a critical application that can be applied within many parts of the industry. Sample-based path planning algorithms, like Rapidly Expanding Random Trees (RRT) have created many spin-offs and optimizations. Bi-Directional RRT* is an optimized sample-based path planning algorithm that searches from both the start and goal points to find an efficient path. in an environment with multiple goals, a Bi-RRT* search algorithm can be used to quickly find and efficient route to move goal to goal, especially when goal permutations are taken into account.**

I. Introduction

According to the 2019 USDA Certified Organic Summary, an estimated 657 farms in the United States produced 831 million pounds of certified organic apples in 2019 with a sales value of 451.2 million dollars. An interesting fact is that each apple sent to the global market, bringing billions of dollars to the nation's economy each year, was manually hand-picked in the orchards. However, The agricultural industry is feeling the strain of labor shortages due to stricter policies limiting the number of migrant workers, rising labor costs, and fewer young people pursuing careers in farming. To combat this problem, many commercial farmers are turning to robots.

Fig. 1. An Autonomous Apple-Picking Robot [19]

The automated apple picking robots are currently capable of picking an apple every 3.6 seconds in a high-density orchard and achieves an 80 percent fruit-picking success rate with minimal bruising or quality issues. Near-optimal path planning for large-scale apple picking robots involves traversing through every apple tree in the orchard, avoiding all possible obstacles, and orienting with respect to the tree trunks such that the end-effectors can retrieve the apples. This is an indispensable part of autonomous apple pickers, as it can generate an asymptotically optimal solution (e.g., in terms of path length, time taken, or energy consumed) in a distinctly cluttered environment.

II. Background

It is important to build the requisite foundation before diving into the intricacies of the Bi-RRT* algorithm. This section provides a basic understanding and enlightens the significance of an effective framework to establish optimality suitable for a large class of problems in domains such as robotics, automation, manufacturing, and computer animation (see [2] and [3]).

A. Optimal Path Planning

Path planning lets an autonomous vehicle or a robot find the shortest (optimal) or most feasible and obstacle-free path from a start to a goal state. The path can be a set of states (position and/or orientation) or waypoints. Provided a map or scaled-map representation of the environment along with start and goal states as input, a path can be planned from the start to the goal state with a variety of approaches. A few popular categories of path-planning algorithms for autonomous vehicles include grid-based search algorithms, sampling-based search algorithms, trajectory optimization algorithms, etc. When it comes to optimal path planning over feasible path planning, the trade-off is between the computational complexity and processing time. As we put robots in a dynamic environment complexities increase, and the complexity of the algorithm also increases with an increase in the degrees of freedom of the robot [4]. This paper discusses further about an improved sample-based path planning approach in the upcoming sections.

B. Sampling Based Algorithms

Sampling based path-planning methods have been developed in the field of robotics for outdoor navigation, proven to be an efficient framework that handles complex problems

in high-dimensional spaces but usually operates in a binary world. The aim is to find out 'near-optimal' collision-free solutions rather than the optimal path according to a cost function, which is usually computed from a model of the terrain [5]. Classical grid-based methods, such as A* or D* [6], can be used to compute resolution-optimal paths over a costmap. However, compared with sampling-based algorithms, these methods are limited to problems that involve low-dimensional spaces that can be discretized and searched using grid-search techniques.

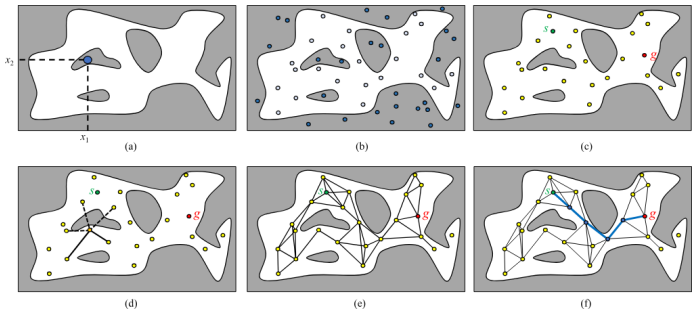

Fig. 2. Example of a Sampling-Based Algorithm in an Obstacle Space [7]

A sampling-based path planner randomly connects points in the state space and constructs a graph to create obstacle-free paths [8]. These algorithms don't require exploring the full configuration space, so they are faster and more efficient. The number of iterations to generate the graph connectivity can be set by the user, which will dictate the optimality of the path that it finds [9]. These types of algorithms present a significant issue while traversing tight spaces, as it is difficult to find connectivity through narrow spaces via random sampling. Probabilistic Roadmap (PRM) and Rapid Exploring Random Trees (RRT) are the two most discussed algorithms in sampling-based path planners, where PRM* and RRT* are the optimized versions of these algorithms [10]. Differentiation in these comes from the way they connect points to create the graph. This paper further expounds on the RRT* and Bi-RRT*, an improvised version of them.

C. RRT

Rapidly exploring random tree (RRT) was developed by Steven M. LaValle and James J. Kuffner Jr. to efficiently search non-convex, high-dimensional spaces by randomly building a space-filling tree [11] [16]. A search-tree is constructed incrementally by randomly drawing samples from the search space (configuration space) and is inherently biased to grow towards large unsearched areas of the problem. RRT algorithm (shown in fig.3) [16] explores the path to the goal while simultaneously creating the graph. First, RRT generates random nodes and checks if the node does not belong to the obstacle space before connecting it to the closest node in the graph. Edges to the closest node should also make sure the path (edge) does not collide with any obstacle. The end condition for this algorithm is that either a complete path is created to a point inside the goal region or the number of iterations has reached its maximum limit. To generate random nodes, any random generator can be used, which will have its own bias, so we can say that it affects the path resulting from RRT. RRT is a fairly quick and easy-to-implement algorithm, but it does not guarantee optimality. It also produces cubic graphs, which are solved by RRT* [12] [13], which we are about to discuss in detail [14].

```
BUILD_RRT(x_init)
1  T.init(x_init);
2  for k = 1 to K do
3      x_rand ← RANDOM_STATE();
4      EXTEND(T, x_rand);
5  Return T
```

```
EXTEND(T, x)
1  x_near ← NEAREST_NEIGHBOR(x, T);
2  if NEW_STATE(x, x_near, x_new, u_new) then
3      T.add_vertex(x_new);
4      T.add_edge(x_near, x_new, u_new);
5      if x_new = x then
6          Return Reached;
7      else
8          Return Advanced;
9  Return Trapped;
```

Fig. 3. RRT Pseudo Code [16]

D. RRT*

Rapidly Exploring Random Tree Star (RRT*), a recently proposed optimized extension of RRT [15], claims to achieve convergence towards the optimal solution, thus ensuring asymptotic optimality along with probabilistic completeness. In addition to the RRT, the new RRT* algorithm calculates the distance to the k-nearest nodes and then calculates the total cost to reach a new node via each neighbor. It chooses the path that corresponds to the minimum cost and connects the nearest neighbor to the new node. Further, it is to rewire the tree to connect the paths (edges) with respect to the least cost. For each neighbor, the cost of traversal through the new node is checked, and if it is less than the best cost obtained earlier, the neighbor's connection to its previous parent is destroyed, and the new node becomes the parent node. RRT* algorithm based on random sampling can provide a collision-free and asymptotic optimal solution for many path planning problems. However, it has been proven to take an infinite time to do so and with a slow convergence rate [17]. RRT* have low sampling efficiency and slow convergence rate in the environment which consists of long corridors, due to a large number of iterations are required in sampling critical nodes. To overcome this problem, this paper adroitly utilises Bi-RRT*,

that introduces extension range, dual-direction exploration, and refinement in trajectory design.

E. Bi-RRT*

Bi-directional rapidly exploring random tree star (Bi-RRT*) is an optimized, computationally cost-effective, and efficiently improved version of the RRT algorithm. Bidirectional search is a search strategy using both start and target points for an initial solution [18]. The Bi-RRT* can be classified into two parts of execution. The first handles the formation of two random trees as we explore new nodes in the configuration space, similar to RRT, simultaneously from the start and goal states. The second part will be the star development of RRT*, which handles optimization by eliminating redundant nodes through node rewiring.

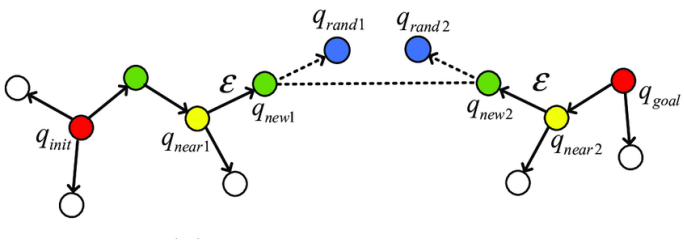

Fig. 4. Bi-RRT* Illustration [20]

The basic principle of the Bi-RRT algorithm is to obtain two random trees, tree1 and tree2, by initializing start and goal state, and the obstacle space. The tree1 extends to the goal state from the start, q_init as the root node. Tree2 extends to the start state form the goal state, q_goal as the root node. Initially, in the global space of the search, the nodes q_rand1 and q_rand2 are selected by random sampling. Provided the randomly selected nodes does not belong to the obstacle space, the new nodes q_near1 and q_near2 are explored by traversing tree1 and tree2 to find the nearest distance from the random nodes. Then, the random nodes are connected with the nearest nodes to determine whether the edge connecting new node and nearest node does not obstruct with any obstacle. If the distance between the nearest node and the random point is equal to the extended extension step size, the nearest random node is added to the tree. On the contrary, the nearest node is connected to the random point. The latest node q_new with a fixed step length is added to the tree, and then the nearest random node is added to the extension repeats the above operation to continue sampling until the distance between the two trees is less than the set threshold. Connect tree1 and tree2, select the sampling point to plan the path [19]. As an critial part optimisation, this paper adroitly deploys the optimisation concept of RRT* algorithm, differentiating from the regular RRT algorithm, to eliminate redundant nodes over each iterations.

Algorithm 1 Bi-RRT* Pseudocode [21]

$V_1 \leftarrow q_{start}; V_2 \leftarrow q_{goal};$
$E_1 \leftarrow 0; E_2 \leftarrow 0;$
$T_1 \leftarrow (q_{start}, E_1); T_2 \leftarrow (q_{goal}, E_2);$
$c_{best} \leftarrow \infty;$
for $iter \in |1, MaxIter|$ **do**
  $q_{rand} \leftarrow Sample(iter)$
  $q_{nearest} \leftarrow Nearst(V_1, q_{rand})$
  $q_new \leftarrow Steer(q_{nearest}, q_{rand})$
  if $CollisionFree(q_{near}, q_{new}, X_{obs})$ **then**
    $X_{near} \leftarrow Near(V_1, q_{new}, R_{near})$
    $q + min \leftarrow ChooseParent(X_{near}, q_{new}, q_{nearest})$
    $q_{near} \leftarrow q_{min}$
    $V_1 \leftarrow V_1 \cup q_{new}$
    $E_1 \leftarrow E_1 \cup (q_{near}, q_{new})$
    $T_1 \leftarrow Rewire(T_1, q_{new}, q_{near})$
  end if
  $q_{connect} \leftarrow Nearest(T_2, q_{new})$
  $(c_{sol}, \sigma_{sol}) \leftarrow ConnectGraphs(T_2, q_{new})$
  if $c_{sol} < c_{best}$ **then**
    $c_{best} \leftarrow c_{sol}$
    $\sigma_{best} \leftarrow \sigma_{sol}$
  end if $Swap(T_1, T_2)$
end for
return $(T_1, T_2) = (V, E)$; =0

III. Method

A. Path Planning Algorithm

The algorithm as described above in pseudo code was implemented into python code, using an RRT package created by the Massachusetts Institute of Technology. This code was thoroughly inspected to ensure that it followed all of the proper steps of Bi-RRT* implementation. There are many parameters within the code that can be tuned, notably the length of each search node, the number of nodes and paths that are rewired, and how often they are rewired. All of these factors play significantly into the computation time of the algorithm, as shorter search paths make the algorithm more likely to take longer to search, but will likely result in less significant overshoots and result in shorter paths.

This planning algorithm will produce waypoints that connect the starting node to each following goal, with intermediate waypoints as calculated by the algorithm. This designed output was chosen because it is adaptable regardless of the robot used, and to compute the goal with a specific mobile robot in mind would cause added difficulty if a different mobile robot is chosen. With waypoints relative to a starting point, and a competent local path planner, any robot can be fed the waypoints to follow the created path.

The Bi-RRT* algorithm used takes advantage of visibility, meaning that if another branch of the goal node is visible, then the algorithm automatically completes the path. This is incredibly beneficial when looking at computation time of the algorithm, though it comes at a cost of efficiency and optimal-

ity. This method will not create a completely optimal path, because even though RRT*, when iterated through multiple times begins to approach optimal, the convergence will take an infinite number of iterations. Despite this, we are analyzing a planning algorithm that will create a feasible and efficient path that will allow an autonomous robot to reach all the goals presented,

B. Cost Function

The cost that was analyzed for this planning algorithm was calculated to be the sum total of the euclidean distance between all of the waypoints. Distance as a cost was chosen due to many considerations, two of which are energy use by the mobile robot, and speed. The shortest path is the most efficient for an autonomous robot.

C. Environment

The simulation used throughout this research project was created through python programming, and included n obstacles, which had corresponding goals assigned to each obstacle. due to the nature of our problem, the apple trees acted as both the obstacles and the goals. The goal points were randomly assigned from 12 pre-designated points surrounding the obstacles. The first obstacle space, "Map 1", contained 5 trees, arranged in an "X" formation. The second obstacle environment, "Map 2" had 6 trees that were arranged in rows. These two maps were created to analyze the time-complexity of n obstacles when evaluating their sequential permutations, as well as to look at the Bi-RRT*s performance when given different formations. Map 2 best represents what a typical apple orchard would have setup as their tree arrangement, as typically the trees are arranged in long rows. The two environments created are as shown below.

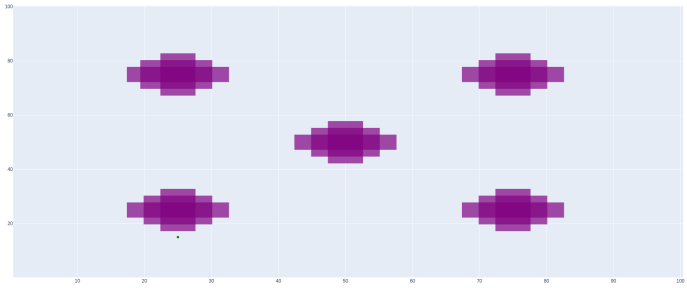

Fig. 5. Map 1: 5 trees aligned in an "X" formation

D. Goal Permutations

with the initial setup of the algorithm and the corresponding goal points for each tree, the algorithm followed each goal sequentially , ending at the final goal. This presents a problem when looking at optimizing the path of the robot movement, since the goals are assigned and ordered randomly, there are many goal orientations that can be disadvantageous for optimal shortest path.

To discover the optimal orientation of the goals relative to the robot, each permutation of the 5 goals was evaluated

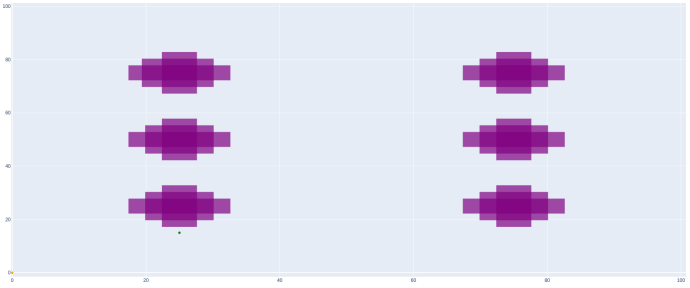

Fig. 6. Map 2: 6 trees aligned in rows

and their cost recorded in order to find the optimal goal permutation for the shortest path. This eliminates the chance that the robot will need to backtrack to a goal it already passed, further increasing the cost of the path.

Permutation search optimization can be very computationally intensive as the number of goals increases, as there are n! permutations for n goals. Our initial method for permutation analysis was to use a brute force search. Each iteration with contain a different permutation of the goals, and for 5 goals, 120 total searches would be executed. This, however, is not feasible for a large-scale operations, and cannot be scaled to larger problems. which was seen with the analysis for Map 2 and the 720 permutations that came with the addition of one extra goal point.

IV. Results

A. Permutation Analysis

When analysing the paths created by the Bi-RRT* search algorithm has an ideal goal arrangement that needs to be found. By iterating through all of the permutations, at a high computational cost, we were able to find a more optimal goal arrangement. Though it is to be noted that there are likely many permutations with similar cost, and due to the random nature of the search, we simply found one of the many ideal goal arrangements.

TABLE I
Permutation Analysis

Map	*Minimum Cost*	*Maximum Cost*	*Computational Time*
Map 1	212.80	399.83	636.79s
Map 2	222.69	452.33	4826.85s

As shown in Table 1, we can see the factorial cost of increasing the number of obstacles and goals with our brute force permutation analysis. This table also shows the value of the permutation analysis, as in Map 1, the maximum cost was 399.83 as compared to the minimum cost of 212.80 and Map 2 we saw an even more stark contrast between the maximum and minimum costs, comparing 452.33 to 222.69.

B. Parameter Tuning

Bi-RRT* Algorithms have a wide variety of parameters associated with the algorithm that can be tuned. The two

significant parameters that were assessed were the rewiring frequency and the branch length. The parameter tuning was only tested on Map 1 and and its corresponding ideal goals.

TABLE II
PARAMETER TUNING

Map	*Rewiring Freq*	*Branch Length*	*Cost*
Map 1	32	8	229.4
Map 1	32	4	211.4
Map 1	32	2	2220
Map 1	16	4	230.2
Map 1	8	4	226.4

Due to the random nature of the search algorithm, a single run is not guaranteed to best represent the performance of the parameters, so each parameter arrangement was tested and the best iteration of 50 tests were used in Table II to represent the performance of each parameter.

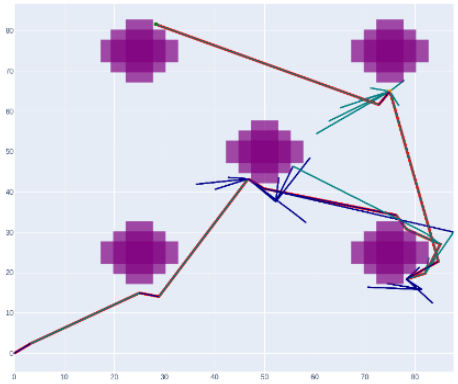

Fig. 7. Map 1 Path Length 4, Rewire Frequency 8

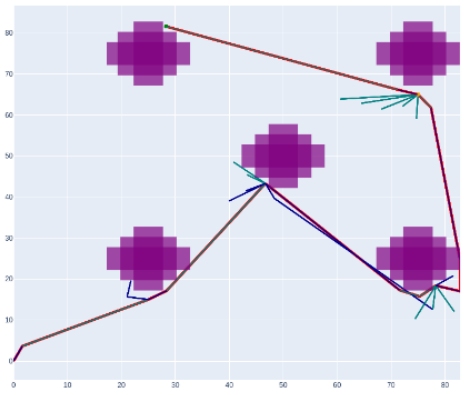

Fig. 8. Map 1 Path Length 4, Rewire Frequency 32

Figures 6 and 7 represent 2 specific parameter test cases. Figure 6 represents a path search that was executed with a branch length of 4 and a rewire parameter of 8. This means that there is a significantly lower requirement of similar branch nodes that are required for the RRT* part of the algorithm to rewire the parent nodes. This can be seen very obviously with very sporadic and numerous branches. On the contrary, Figure 7 displays a rewire frequency of 32, which we found to be the most optimal through our simulations. There are significantly less branches, and they appear to be much more focused in the area of the search, which allows for an overall more efficient and concise search

Following the tuning of these parameters, that results were able to determine that the ideal branch length and rewiring frequency for the environment was a branch length of 4 meters, while rewiring every 32 branches. Additionally, the analysis found no significant difference in computational time between any combination of parameters.

C. Simulations

Following the permutation analysis and parameter tuning, the Bi-RRT* algorithm was run 100 times in order to find the most ideal path in terms of cost. This is required as the creation and selection of branches is completely random, and even if we have found an ideal goal order to search, we must still find iterate through numerous simulations to find the best performance of the algorithm within the constraints.

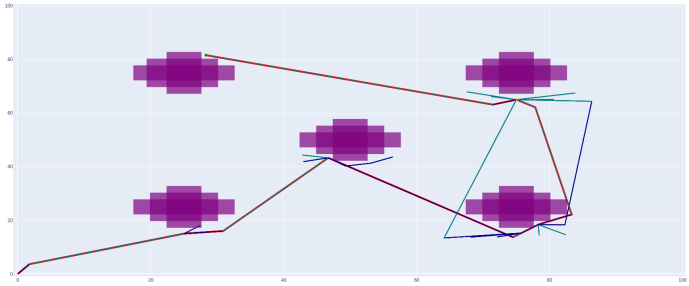

Fig. 9. Map 1 path with Bi-RRT* Algorithm

Figure 4 shows the path as created by the Bi-RRT* algorithm for map 1. The final cost for this final path is 209.9 meters after 100 iterations. While there are clearly areas for further optimizations, such as branches that can be condensed down into straight lines, this is still a feasible and efficient path. A notable difference in this path compared to many iterations that were run for this goal set is that the 3rd obstacle, in the bottom right of the map often had paths generated that went over the top of the obstacle and then double backed on itself, creating inefficiencies for the path traveled by the robot.

Figure 5 shows the path as created by the Bi-RRT* algorithm for map 2. The final cost for this final path is 211.74 meters. As with the Bi-RRT* performance on map 1, this performance is not optimal, but it is feasible. Non-optimal paths can be seen around the goal to the left of obstacle 4 to the top right of the map, where the path seemingly takes a sharp turn in the opposite direction of the next goal.

V. CONCLUSION

This project has successfully created a path-planning pipeline to create a feasible path for a robot to travel throughout an apple orchard while targeting variable goals and orien-

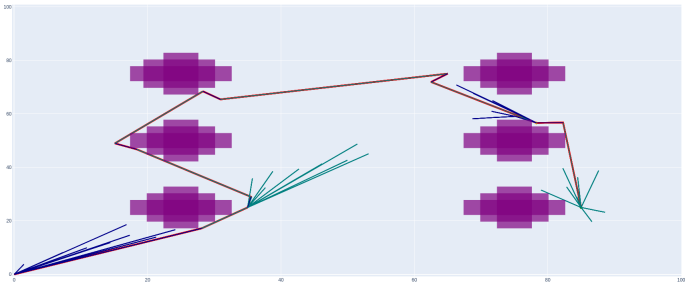

Fig. 10. Map 2 path with Bi-RRT* Algorithn

tations. The Bi-RRT* algorithm displayed that it was able to compute a feasible path through a sampling-based algorithm within 5-6 seconds, depending on the number of goals. For small scale, unknown applications, this whole pipeline can be used to quickly create a feasible path for an autonomous agricultural robot. If goal permutations are not included, the Bi-RRT* algorithm can be used to very rapidly compute a feasible path for any size or scale of operation, as long as the goal order is known.

The planning algorithm. specifically the optimal permutation analysis, does struggle with time complexity with increasing obstacles, as the permutation analysis relies on a brute-force search, and there are many ways to optimize the ordering of goal waypoints.

REFERENCES

[1] "Apple harvesting robot plucks a piece of fruit every 7 seconds," New Atlas, Apr. 28, 2021. https://newatlas.com/robotics/apple-harvesting-robot-fresh-seven-seconds/

[2] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, et al., Principles of Robot Motion: Theory Algorithms and Implementations, MA, Cambridge:MIT Press, 2005.

[3] S. LaValle, Planning Algorithms, New York:Cambridge Univ. Press, 2006.

[4] Karur, K., Sharma, N., Dharmatti, C., and Siegel, J. E. A survey of path planning algorithms for mobile robots. Vehicles, 3(3):448–468, 2021.

[5] L. Jaillet, J. Cortés and T. Siméon, "Sampling-Based Path Planning on Configuration-Space Costmaps," in IEEE Transactions on Robotics, vol. 26, no. 4, pp. 635-646, Aug. 2010.

[6] A. Stentz, "Optimal and efficient path planning for partially-known environments", Proc. IEEE Int. Conf. Robot. Autom., pp. 3310-3317, 1994.

[7] "MotionPlanningHigherDimensions," motion.cs.illinois.edu. https://motion.cs.illinois.edu/RoboticSystems/MotionPlanningHigherDimensions.html

[8] Karaman, S. and Frazzoli, E. "Sampling-based algorithms for optimal motion planning." The International Journal of Robotics Research, 30(7):846–894, 2011a.

[9] Marin-Plaza, P., Hussein, A., Martin, D., and Escalera, A. d. l. Global and local path planning study in a ros-based research platform for autonomous vehicles. Journal of Advanced Transportation, 2018:1–10, 2018.

[10] Alka Choudhary, "Sampling-based Path Planning Algorithms: A Survey", arXiv:2304.14839v1 [cs.RO] 23 Apr 2023.

[11] LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning". Technical Report (TR 98–11). Computer Science Department, Iowa State University, Oct. 1998.

[12] Karaman, S. and Frazzoli, E. "Sampling-based algorithms

[13] Jia Zhu, Shili Zhao, Ran Zhao, "Path Planning for Autonomous Underwater Vehicle Based on Artificial Potential Field and Modified RRT", International Conference on Computer, Control and Robotics, 2021.

[14] OlzhasAdi (Jan 26, 2015). "RRT* Brief Explanation" (video). YouTube. Archived from the original on 2021-12-12. Retrieved 3 August 2016.

[15] Karaman, Sertac; Frazzoli, Emilio., "Incremental Sampling-based Algorithms for Optimal Motion Planning", May, 2013.

[16] Steven M. Lavalle, James J. Kuffner, Jr. "Rapidly-Exploring Random Trees: Progress and Prospects" Algorithmic and Computational Robotics: New Directions, 2000.

[17] JOUR., Nasir, Jauwairia., Islam, Fahad., Malik, Usman Ali., Ayaz, Yasar., Hasan, Osman., Khan, Mushtaq., Muhammad, Mannan., "RRT*-Smart: A rapid convergence implementation of RRT*", International Journal of Advanced Robotic Systems, Jan. 2013.

[18] Fan H, Huang J, Huang X, Zhu H, Su H. "BI-RRT*: An improved path planning algorithm for secure and trustworthy mobile robots systems.", Feb, 2015.

[19] Tianhao Qin and Shaojie Xin 2022 J."Path planning algorithm based on improved Bidirectional RRT", Phys.: Conf. Ser. 2396 012055.

[20] Sun, Jian Zhao, Jie Hu, Xiaoyang Gao, Hongwei Yu, Jiahui. (2023). Autonomous Navigation System of Indoor Mobile Robots Using 2D Lidar. Mathematics. 11. 1455. 10.3390/math11061455.

[21] Jiaming Fan, Xia Chen, Xiao Liang, UAV trajectory planning based on bi-directional APF-RRT* algorithm with goal-biased,"Expert Systems with Applications, Volume 213, Part C, 2023, 119137, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2022.119137. for optimal motion planning." The international journal of robotics research, 30(7):846–894, 2011b.