A TD-RRT* Based Real-Time Path Planning of a Nonholonomic Mobile Robot and Path Smoothening Technique Using Catmull-Rom Interpolation

Sivaram Dheeraj Vishnubhotla
A. James Clark School of Engineering
(University of Maryland, College Park)
College Park, Maryland, USA
sivaram@umd.edu

Alexander Charles Hall
A. James Clark School of Engineering
(University of Maryland, College Park)
College Park, Maryland, USA
ahall113@umd.edu

***Abstract*—In the subject of mobile robotics, the adept planning of an optimal trajectory from the robot's initial position to a goal position is a fundamental necessity. While this task may seem trivial in static environments, the operational setting for such robots is invariably dynamic, often characterized by many dynamic obstacles. These dynamic obstacles, unforeseen and variable, pose potential impediments to the robot's trajectory. Consequently, the robot must dynamically adapt its path when encountering such obstacles, ensuring uninterrupted progression towards its goal.**

This paper aims to establish a cogent relationship among the parameters employed in path-planning algorithms, facilitating the seamless movement of robots within dynamic environments. Specifically, the aim is to devise a platform wherein a robot can navigate from its starting point to a defined destination amidst both stationary and mobile obstacles, without collision. To achieve this objective, we adopt an asymptotically optimal variant of the Rapidly-exploring Random Tree (RRT) algorithm, known as RRT*. Rooted in incremental sampling, this algorithm comprehensively explores the configuration space while exhibiting swift computational performance, rendering it suitable for deployment in multidimensional environments.

Furthermore, we introduce a method of dynamic replanning utilizing Temporal Difference RRT* (TD-RRT*). This approach enables the robot to recalibrate its trajectory in response to random and unforeseen obstacles encountered during traversal. Experimental evaluations underscore the efficacy of the proposed methodology, demonstrating its superior speed compared to the baseline RRT*, while concurrently yielding smoother trajectories adhering to the nonholonomic constraints inherent to mobile robots.

***Index Terms*—Path planning, Non-holonomic mobile robot, Rapidly-exploring random trees, Triangle decomposition**

I. Introduction

Mobile robots represent a subset of intelligent vehicles widely applied across various sectors, demonstrating significant real-world benefits. Among their core technologies, path planning is crucial, especially given current challenges in devising methods that can effectively navigate complex environments while addressing practical constraints such as vehicular limitations and the necessity for real-time dynamic responses.

Rapidly exploring random trees (RRT), a respected strategy in robotic navigation utilizes a single-query, sampling-based approach that expands a configuration tree to progressively encompass the entire state space. The RRT* algorithm, an evolution of RRT, seeks to refine this approach by optimizing the path-finding process. Research has introduced numerous enhancements to RRT, notably a variant termed RRT* that prioritizes optimality, clearance, and completeness in path planning. These improvements focus on ensuring paths are optimal, reduce collision risks, and guarantee path discovery if one exists.

As industries evolve, the next-generation factories demand mobility and adaptability, driving the integration of intelligent mobile robots. Recent advances in machine learning and artificial intelligence have further propelled this shift towards automation. Effective path planning for mobile robots must now meet the demands of kinematics, dynamics, and environmental interactions. Within this framework, RRT* has been adapted through additional functionalities to generate smoother, safer paths. These adaptations include integrating motion primitives, optimal control theories, and spline techniques to enhance the algorithm's ability to identify collision-free trajectories and address local planning challenges.

The literature on RRT and its derivatives is extensive, highlighting attributes such as probabilistic completeness and the diminishing likelihood of failure. These methods have proven effective in systems characterized by differential constraints, nonlinear dynamics, and various constraint types, including non-holonomic and discrete or hybrid systems.

This paper explores motion planning for nonholonomic mobile robots in two-dimensional, real-time complex environments. By employing novel extensions to the standard RRT* framework, such as target bias strategies, triangle decomposition, and effective metric functions, the study aims to enhance path smoothness and navigational efficacy in human-centric settings.

II. Overview of the Framework

A. *RRT* Algorithm*

Algorithm 1 The basic pseudo-code for RRT* algorithm is given below:

```
1:  $V \leftarrow x_{\text{start}}$; $E \leftarrow \emptyset$; $T \leftarrow (V, E)$
2:  while NotFindStop do
3:      $x_{\text{rand}} \leftarrow$ Sample_free
4:      $x_{\text{nearest}} \leftarrow$ Nearest($x_{\text{rand}}, T$)
5:      if Collision_check($x_{\text{nearest}}, x_{\text{new}}$) then
6:          continue
7:      end if
8:      $T \leftarrow$ InsertNode($x_{\text{new}}$)
9:      $T \leftarrow$ Rewire($x_{\text{new}}, T$)
10:     if Goal_find($x_{\text{goal}}, T$) then
11:         NotFindStop $\leftarrow$ False
12:     end if
13: end while=0
```

Rapidly Exploring Random Trees (RRT) is a sampling-based path-planning algorithm that can efficiently explore a defined high-dimensional space. In simple terms, a goal location can be quickly found even in a large area. The idea is that if enough iterations of the algorithm are executed then a goal location will eventually be found. The downside is the path produced from the RRT algorithm is usually far from optimal and there can be a lot of variance in the path produced.
The RRT algorithm works by sampling a random point in the map space. The nearest node to the newly sampled point is then located. The two nodes can be directly connected or more ideally, kinematics and step size will be used to move from the near node toward the sampled node and this point will become the new node. If there is no obstacle located between the two nodes it is then added to the tree. The process continues for the defined number of iterations and/or until a goal node is found. Backtracking can then be completed to find the path from the start node to the goal node.

To introduce some optimization to the RRT algorithm, RRT-Star (RRT*) was created. RRT* has two key factors that improve the RRT algorithm. The first factor is RRT* does not directly connect the new node to the nearest node, instead a radius around the new node is chosen and all nodes that are within that area are evaluated for the cost to come. The node that lies within the near threshold and has the lowest cost to come will be the connected node. The second factor is rewiring, which is completed similarly to the first factor. Nodes are evaluated within a defined threshold for the lowest cost to come and the node of interest will be rewired if a lower cost is found. These two factors that improve the tree locally during the algorithm ultimately produce a tree that will include a path more optimal than RRT alone.

Basic understanding of RRT and RRT*:

- RRT is a single-query sampling-based planner that grows a tree of configurations to eventually cover the entire state space.
- RRT* trees grow according to the notion of a cost: under the assumptions, the solution converges to the optimum as the number of samples approaches infinity.

Working of RRT* [1]:

1) RRT* first records the distance each vertex has traveled relative to its parent which is referred to as the cost of the vertex.
2) After the closest node is found on the map, a neighborhood of vertices in a fixed radius from the new node is examined.
3) If a node with the least cost or cheaper cost than the adjacent node is found then the cheaper node replaces the adjacent node.
4) Once the vertex has been connected to the cheapest neighbor, the neighbors are examined again just to make sure that the rewired neighbor will make any cost decrement, and if the cost decreases the neighbor is rewired to the newly added vertex.

B. *Non-Holonomic Motion of Mobile Robots*

Non-holonomic constraints define limitations on the motion of mobile robots, where certain movements are not permissible despite the absence of constraints on the configuration space.

Consider the case of a mobile robot designed for complex environmental interaction. The robot's configuration is defined by the generalized coordinates $q = (x, y, \theta)$, representing its position and orientation in a plane. Here, x and y denote the Cartesian coordinates, and θ is the heading angle.

The robot's motion is characterized by non-holonomic constraints because it can move forward or backward but cannot slide perpendicular to its heading direction without rotating. This limitation is crucial in path planning, especially in real-time applications where the robot must navigate through obstacles while maintaining a feasible trajectory.

The angular velocity of the robot, as part of its motion dynamics, governs how the reference point on the robot tracks a circular path, emphasizing the influence of non-holonomic constraints on its navigational capabilities. These dynamics are critical in developing advanced algorithms for path planning and control that accommodate the inherent physical constraints of the robot.

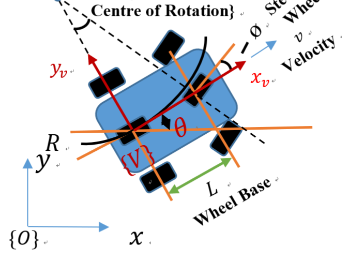

Fig. 1. Non Holonomic mobile robot [1]

C. Triangle Decomposition

To improve the basic RRT* path planning algorithm, triangle decomposition can be implemented, known as TD-RRT*, to further optimize the path produced by RRT*. This works by eliminating unnecessary intermediate nodes, thus reducing ineffective robot movement along the path. This basis of triangular decomposition is derived from the triangular inequality theorem. This theorem simply states that for a triangle, the sum of the length of two sides will be greater than the length of the third side. This can be looked at in the context of mobile robot path planning in Figure X. Consider the vertices of the triangle, A, B, C, as nodes on the path and the edges are the paths to be traveled, it is shorter and more efficient to travel from node A to node C directly then to travel from node A to node B then to node C. If no obstacle lies between node A and node C, then the path can be rewired to this, and node B is eliminated from the path.

Triangle decomposition is applied to the path that is produced from the RRT* algorithm. The triangle decomposition function works by looping through nodes on the generated path. Starting from the goal node and working towards the start node, connects to parents and older generations of parents are made until the obstacle function check is true, which means the current node and last iteration node before the obstacle check are the shortest decomposed leg to the obstacle. The new current node is the last end node, and the process repeats until the start node of the path is reached. It is important to note that the tree that is produced from the RRT* algorithm is not altered or improved through this triangle decomposition function. The triangle decomposition function adds little computation time but improves the cost of the path substantially.

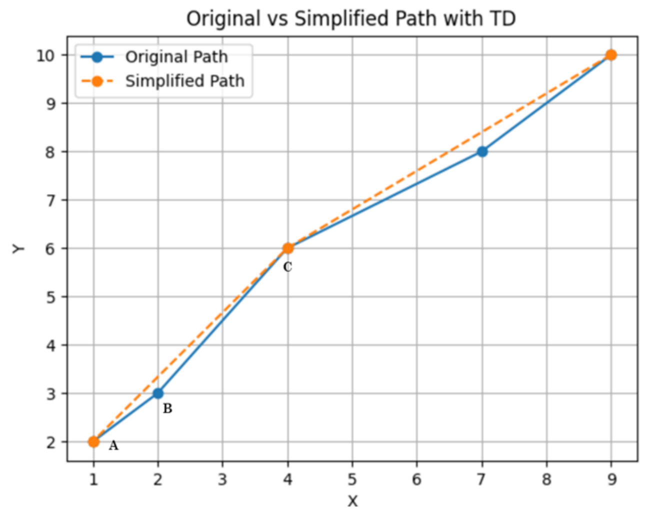

Fig. 2. Triangle Decomposition

In researching triangle decomposition, some RRT* variants apply this concept locally when adding new nodes, different from the method being implemented here within that focuses just on the final path. One method is referred to as Triangular Inequality Based RRT-Connect Algorithm. When a node is inserted and would typically be connected to a parent based on the defined cost-to-come criteria, the next ancestor (grandparent) is connected directly to the new node and checked for obstacle interference. If no obstacle is in between the child is connected to the grandparent instead. This method improves the overall tree that is constructed. Unfortunately, this added checking at each iteration would increase computation time and would likely be greater than that of the TD function applied to just the path. Future work would be to implement this logic into the RRT* algorithm and compare to TD-RRT* and compare computation time and path cost to evaluate which method is more optimal

D. Catmull-Rom Spline Interpolation

In general mobile robots, due to the nature of being non-holonomic, are limited in their ability to take sharp turns. As mentioned above, a typical mobile robot cannot move directly left or right without also moving forward are backward. This makes it difficult for mobile robots to follow small degrees of curvature. To combat this, path smoothing can be applied to the TD-RRT* algorithm.

Catmull-Rom Spline interpolation is a method that will be used to smooth the path that is produced from the TD-RRT* algorithm. The basis of the function is to connect points through splines which are essentially smooth curves as shown in Figure 3 . Using nodes from the path as fixed points, intermediate points are added between with uniform spacing. The interpolation function allows the degree of curvature to be adjusted to fit the needs of a particular mobile robot through the tensioning coefficient. The result is a path that connects nodes through smooth changes of direction that are suitable for a mobile robot to follow.

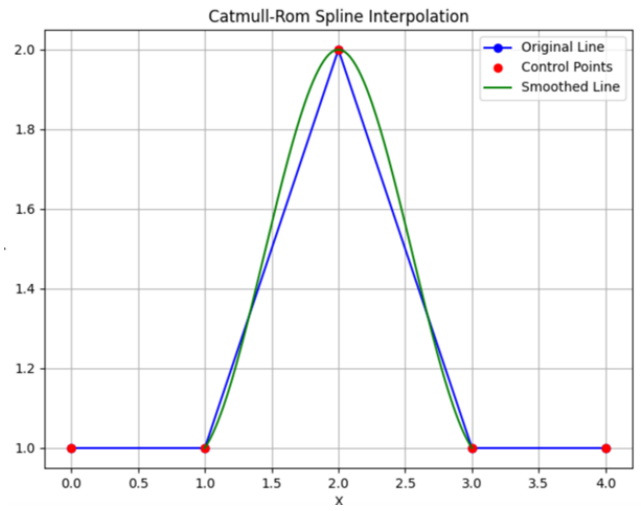

Fig. 3. Catmull-Rom Splines

Interpolated points are defined by Equation 1. Nodes are extracted from the path and treated as control points for the function, x and y coordinates are handled independently. The degree of curvature is set to 0.5 as it is in the referenced research paper [1]. The t value is adjusted from 0 to 1 with equal spacing to create interpolated points, the number of points can be adjusted depending on the required resolution.

The equation is given by:

$$q(t) = 0.5\left((2P_1) + (-P_0 + P_2)t + (2P_0 - 5P_1 + 4P_2 - P_3)t^2 + (-P_0 + 3P_1 - 3P_2 + P_3)t^3\right)$$

Algorithm 2 TD-RRT* with Smoothening function

```
1: $x_{\text{start}} \leftarrow (x, y)$
2: $x_{\text{goal}} \leftarrow (x, y)$
3: $max\_iter \leftarrow$ number of iterations
4: $iter\_count \leftarrow 0$
5: $T \leftarrow$ initialize tree with $x_{\text{start}}$
6: while $iter\_count < max\_iter$ do
7:     $x_{\text{rand}} \leftarrow$ sample_free(map_size) {random x,y point}
8:     $near\_nodes \leftarrow$ find_near_nodes$(x_{\text{rand}}, T, max\_distance)$
9:     $x_{\text{nearest}} \leftarrow$ find_parent_in_threshold$(x_{\text{rand}}, near\_nodes)$
10:    $x_{\text{rand\_new}} \leftarrow$ move_towards$(x_{\text{nearest}}, x_{\text{rand}}, step\_size)$
11:    if collision_check$(x_{\text{nearest}}, x_{\text{rand\_new}})$ then
12:        continue
13:    end if
14:    $T \leftarrow$ insert_node$(x_{\text{rand\_new}}, T)$
15:    $current\_cost\_segment \leftarrow$ c2c$(x_{\text{nearest}}, x_{\text{rand\_new}})$
16:    $current\_cost\_node \leftarrow current\_cost\_segment +$ cost_total$[x_{\text{nearest}}]$
17:    node_tracker.append$((x_{\text{nearest}}, x_{\text{rand\_new}}, current\_cost\_node))$
18:    $T \leftarrow$ rewire$(x_{\text{rand\_new}}, T)$
19:    if goal_find$(x_{\text{goal}}, x_{\text{rand\_new}})$ then
20:        $back\_track\_start \leftarrow x_{\text{rand\_new}}$
21:    end if
22:    $iter\_count \leftarrow iter\_count + 1$
23: end while
24: $original\_Path \leftarrow$ backtrack$(T, x_{\text{rand\_new}}, x_{\text{start}})$
25: $T_{\text{Triangle\_Decomp}} \leftarrow$ td$(original\_Path)$
26: $T_{\text{Smooth}} \leftarrow$ smoothing$(T_{\text{Triangle\_Decomp}})$ =0
```

III. Results and graphs

All simulations were executed in a map space that was representative of the referenced research paper [1]. Static obstacles were located and sized the same, dynamic obstacles were sized the same but kinematics varied since they were not provided. The size of the map space, 800 x 800, was reproduced without adjusting the scale. The start and goal nodes selected were chosen to match that of the referenced research paper [1]. The number of algorithm iterations was not provided in the reference research paper [1].

Many simulations were executed, the results of 5 are shown in the figures below [Fig 4 - 8] with numerical results in the table below [Tables 1 and 2]. The number of iterations was adjusted for the simulations ranging from 500 to 6000. Focusing on a single run, it is apparent that the path cost is improved from basic RRT* to TD-RRT*. The cost was reduced from 15-20 percent for the sample runs. That is a significant improvement and given that the computation time increase was only 50-100 ms, applying the Triangle Decomposition function to the algorithm proves to be valuable. The cost of the Smoothed path was estimated using Euclidean distance between nodes so the actual cost will be slightly higher than the values provided but still improved from RRT*. The path smoothing also proved to be valuable as cost is not sacrificed and an optimal trajectory can be prescribed.

There were both pros and cons observed to increasing the iterations for the complete algorithm. Generally speaking, by increasing the iterations the goal was more likely to be found, the downside is that the computation time increases drastically. An appropriate middle ground for the map was somewhere between 2000-3000 iterations. The goal was found a majority of the time, an optimized path was produced, and the computation time was relatively low. The cost-to-iteration relationship did not appear to have a strong correlation. Whenever a goal was found, even with low iterations, the cost was comparable to that of high iterations. Many more samples would have to be run for each iteration to determine a statistically significant correlation between the two variables. The below tables show the path cost and computation time comparing RRT*, TD-RRT*, and smoothened RRT*:

TABLE I
Path Cost

Run Sample	Number of Iterations	RRT*	TD-RRT*	Smoothed RRT*
1	500	824.851	646.315	540.154
2	1000	685.946	573.480	530.831
3	2000	936.644	742.441	736.954
4	4000	745.171	630.591	548.910
5	6000	895.947	683.531	595.236

TABLE II
Computation Time (seconds)

Run Sample	Number of Iterations	RRT*	TD-RRT*	Smoothed RRT*
1	500	4.577	4.624	4.625
2	1000	3.991	4.022	4.023
3	2000	12.981	13.028	13.029
4	4000	48.877	48.924	48.925
5	6000	115.977	116.074	116.076

As the number of iterations increases we can see how the path from the start node to the goal node gets smoother to achieve the fastest path possible according to the robot dynamics. The below figures show how the path planned from the start node to the goal node using TD-RRT* with the application of smoothening functions at different steps of iterations:

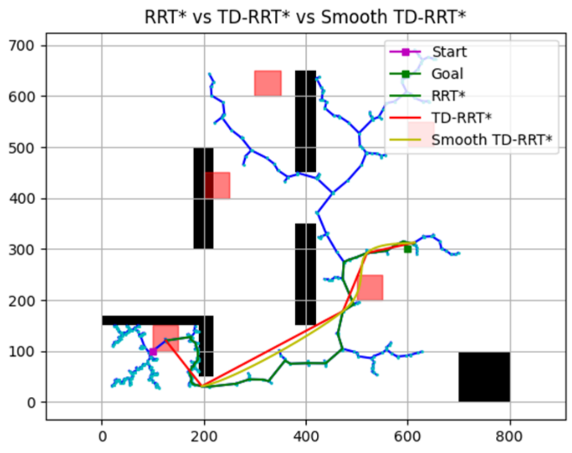

Fig. 4. 500 iterations

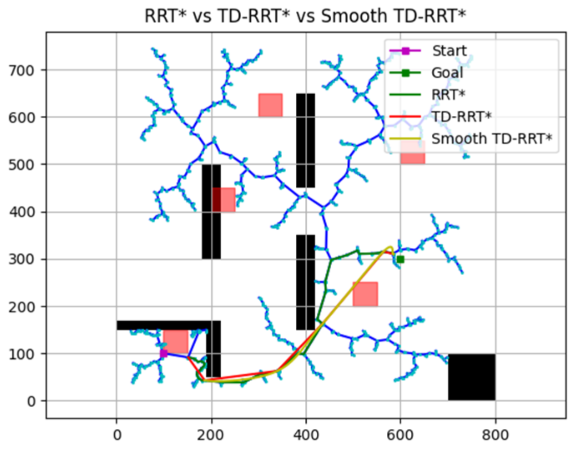

Fig. 5. 1000 iterations

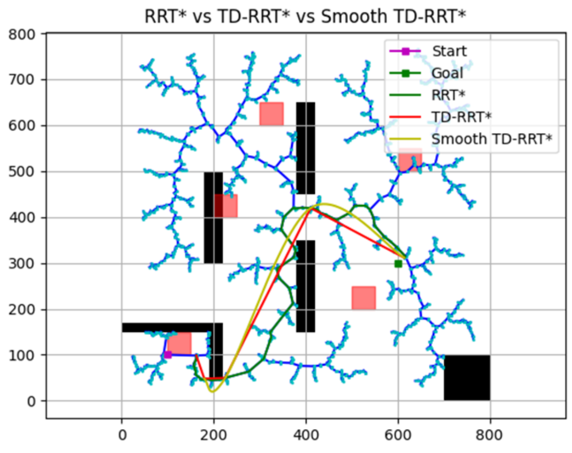

Fig. 6. 2000 iterations

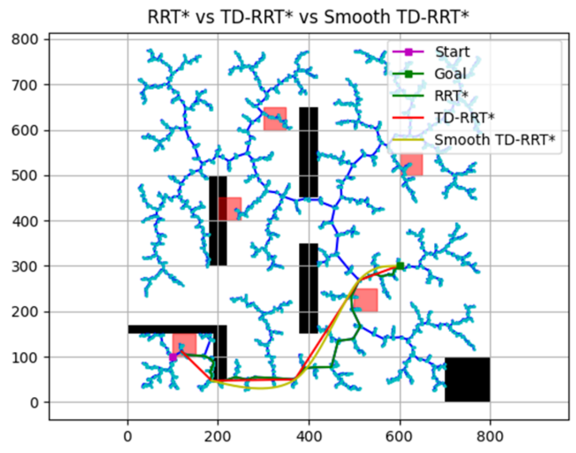

Fig. 7. 4000 iterations

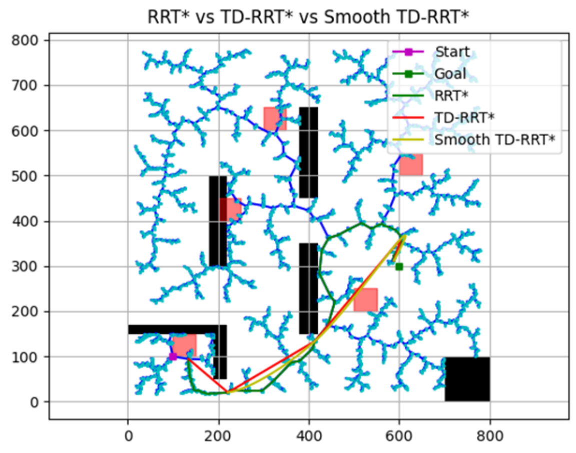

Fig. 8. 6000 iterations

IV. Conclusion

The integration of Triangle Decomposition (TD) with the Rapidly-exploring Random Tree Star (RRT*) algorithm significantly enhances robotic path planning capabilities. This method retains the core advantage of RRT*, which is the rapid exploration of the environment, and complements it with an effective optimization strategy for pathfinding. Through the implementation of TD, the algorithm achieves a more targeted exploration and efficient path shortening, which reduces the travel distance from the start to the goal.

Although incorporating TD incurs a slight increase in computational time, the improvement in path quality is substantial. By strategically shortening the path, the time required for a mobile robot to reach its destination is considerably decreased. This optimization not only enhances the robot's operational efficiency but also conserves energy resources and increases its performance speed, which are crucial attributes in time-sensitive applications.

Moreover, the application of interpolation techniques, such as Catmull-Rom smoothing, refines the trajectory that the mobile robot follows. This smoothing process produces paths that are optimal not only in terms of length but also in maneuverability. The smoothed paths enable mobile robots to navigate around both static and dynamic obstacles with greater agility and safety, enhancing their functionality in complex environments.

In conclusion, the augmented RRT* algorithm, enriched by Triangle Decomposition and interpolation smoothing, substantially advances the path-planning processes of mobile robots. This approach provides a robust solution to the navigation challenges in diverse applications, ranging from automated warehousing and urban transportation to robotic operations in hazardous environments. The results highlight the potential of sophisticated path-planning algorithms to revolutionize the field of robotics by facilitating more dynamic and responsive navigation strategies.

V. Future Work

Now that results have been captured from the TD-RRT* algorithm in Python simulations, some more work can be done to further expand on the validity of these results. The algorithm should be executed with simulation software such as ROS/Gazebo after kinematics constraints are applied. The kinematics for the robot being simulated will be incorporated into the move towards function of the algorithm and the tensioning coefficient of the path smoothing function must be tuned accordingly. A simulation can then be ran with both static and dynamic obstacle to assess whether the algorithm is still viable.

Additionally, a comparison should be done with other improved methods of RRT* such as RRT*- Smart, Quick-RRT*, and RRT*-Connect. The computation time and cost of the produced path should be compared to decide which algorithm best suits a mobile robot faced with static and dynamic constraints. Depending on the environment (obstacle type, robot type, map dimension, etc.), certain RRT* variants may be more optimal. So, caution must be given when stating one method definitively outperforms the other.

Efforts should be focused on improving the TD-RRT* algorithm itself, some parameters can be adjusted to find the optimal settings for the environment. Iterations of random sampling need to be further assessed to determine what is the smallest amount that produces acceptable optimal results, the same goes for the size of the threshold radius for node-connect checking and rewiring. Increasing the values of these variables increases the computation time but also aides in improving the optimality, so finding the ideal combination of these could prove valuable in improving this algorithm.

References

[1] Jyotish and Mei-Yung Chen, "A TD-RRT* Based Real-Time Path Planning of a Nonholonomic Mobile Robot and Path Smoothening Technique Using Catmull-Rom Interpolation," in *2022 International Conference on System Science and Engineering (ICSSE)*, 2022, pp. 115-120, doi: 10.1109/ICSSE55923.2022.9948258.

[2] "D. Rose", *Interpolation — Dances with Code*, 2022. [Online]. Available: https://danceswithcode.net/engineeringnotes/interpolation/interpolation.html. [Accessed: May 9, 2024].

[3] Ed Catmull, "Subdivision for Modeling and Animation," Microsoft DirectX Developer Center, [Online]. Available: https://www.mvps.org/directx/articles/catmull/

[4] J. Nasir, F. Islam, U. Malik, et al., "RRT*-SMART: A Rapid Convergence Implementation of RRT*," *International Journal of Advanced Robotic Systems*, vol. 10, no. 7, 2013, doi:10.5772/56718.

[5] Jin-Gu Kang, Dong-Woo Lim, Yong-Sik Choi, Woo-Jin Jang, and Jin-Woo Jung, "Improved RRT-Connect Algorithm Based on Triangular Inequality for Robot Path Planning," *Sensors*, vol. 21, no. 2, p. 333, Jan. 2021. doi: 10.3390/s21020333.

[6] J.-G. Kang, D.-W. Lim, Y.-S. Choi, W.-J. Jang and J.-W Jung, "Improved RRT-Connect Algorithm Based on Triangular Inequality for Robot Path Planning", *Sensors*, vol. 21, no. 333, 2021, [Online]. Available: https://doi.org/10.3390/s21020333.

[7] S. M. Lavalle and J. J. Kuffner, "Rapidly-exploring random trees: progress and prospects," Proceedings of the Workshop on the Algorithmic Foundations of Robotics, pp. 293-308, 2000.

[8] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," The International Journal of Robotics Research, vol. 20, no. 5, pp. 378-400, 2001.

[9] X. Tang, J. M. Lien, and N. M. Amato, "An obstacle-based rapidly-exploring random tree," Proceedings 2006 IEEE International Conference on Robotics and Automation, vol. 2006, pp. 895-900, 2006.

[10] F. Wu and Z. Y. Geng, "Coordinating control of multiple rigid bodies based on motion primitives," Acta Mechanica Sinica, vol. 28, no. 2, pp. 482-489, 2012.

[11] F. Gómez-Bravo, F. Cuesta, A. Ollero, et al., "Continuous curvature path generation based on -spline curves for parking manoeuvres," Robotics and autonomous systems, vol. 56, no. 4, pp. 360-372, 2008.

[12] M. Elbanhawi, M. Simic, and R. Jazar, "Continuous-curvature bounded trajectory planning using parametric splines," Frontiers in artificial intelligence and applications, vol. 262, pp. 513-522, 2014.

[13] J. Cortes, L. Jailet, and T. Simeon, "Molecular disassembly with RRT like algorithms," IEEE International Conference on Robotics and Automation (ICRA), 2007.

[14] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," International Journal of Robotics Research, vol. 20, no. 5, pp. 378-400, May 2001.

[15] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "Rrt*-smart: Rapid Convergence Implementation of rrt* towards Optimal Solution," Proceedings of the IEEE International Conference on Mechatronics and Automation, pp. 5-8, August 2012.

[16] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, "Quick-RRT*: Triangular inequality based implementation of RRT* with improved initial solution and convergence rate," Expert Syst., pp. 82-90, 2019.

[17] J.-G. Kang, D.-W. Lim, Y.-S. Choi, W.-J. Jang, and J.-W. Jung, "Improved RRT-Connect Algorithm Based on Triangular Inequality for Robot Path Planning," Sensors, vol. 21, no. 333, 2021, [online] Available: https://doi.org/10.3390/s21020333.