RT-RRT*- A REAL TIME PATH PLANNING ALGORITHM

SAI DINESH GELAM
University of Maryland
College Park, United States
sgelam@umd.edu

GOWTHAM VMS CHINTALAPATI
University of Maryland
College Park, United States
gowch15@umd.edu

***Abstract*— This technical study introduces the Real-Time Rapidly Exploring Random Trees (RT-RRT*) algorithm, a pioneering iteration of the widely utilized Rapidly Exploring Random Tree (RRT*) approach tailored specifically for dynamic scenarios. Unlike its predecessors, RT-RRT* incorporates a dynamic online tree rewiring technique that continuously updates the path planning tree in real-time, responding adeptly to fluctuations in the agent's position and environmental obstacles. This innovative adaptation enables RT-RRT* to excel in navigating complex and unpredictable environments, a common challenge encountered in autonomous robotics and real-time simulation applications. By prioritizing localized path planning adjustments over global path planning modifications, RT-RRT* enhances its adaptability and efficacy in dynamically changing settings. Extensive simulation experiments conducted to evaluate RT-RRT*'s performance demonstrate its superiority over conventional path planning algorithms in terms of both efficiency and flexibility. The study concludes by advocating for the adoption of localized path planning strategies, emphasizing their effectiveness in navigating dynamic environments with unparalleled efficiency.**

I. INTRODUCTION

Autonomous navigation in dynamic situations is one of the most demanding areas of robotics and artificial intelligence. Systems built for such environments must not only identify and respond to static impediments, but also adapt to unexpected changes such as moving objects and changing endpoints. Traditional path planning algorithms, while effective in predictable situations, frequently fail to fulfill the real-time demands imposed by dynamic environments due to their inability to quickly adjust trajectories as conditions change. This constraint is especially important in applications like driver-less vehicles, where the ability to immediately adapt to a changing environment might be a question of safety. Similarly, in robotic search and rescue missions, the effectiveness and timeliness of reaction can have a substantial impact on emergency outcomes. Effective path planning is thus critical for autonomous systems' operational performance, especially in complex and dynamically changing contexts like autonomous vehicle navigation and robotic disaster response. The RT-RRT* algorithm overcomes these constraints by incorporating a unique tree rewiring process into the established RRT* architecture, allowing for continual adaptation of the path planning strategy as new environmental information becomes available. This article describes the RT-RRT* algorithm, its theoretical foundations, and practical applications using simulation data. This approach systematically contrasts the efficacy of both localized and general path planning methodologies, ultimately affirming the superior efficiency of localized path planning within dynamically changing environments.

II. BACKGROUND AND LITERATURE REVIEW

A. Background

Before getting into Real-Time Rapidly-exploring Random Trees (RT-RRT*), it is imperative to delve into the foundational concepts of Rapidly-exploring Random Trees (RRT) and their various adaptations. Here's a detailed look into the variants of RRT, providing a broad perspective on how the algorithm has evolved to tackle different challenges in path planning.

RRT: Introduced by Steven M. LaValle in 1998, the basic RRT algorithm quickly became a popular choice for path planning in spaces with high dimensional complexity. RRT constructs a tree rooted in the starting configuration by randomly sampling the space. The tree grows towards these samples, forming a path to the target. RRT's strength is its ability to effectively cover the search space, making it appropriate for contexts where other approaches may struggle owing to dimensionality or obstacle configurations.

RRT*: Developed to overcome the non-optimality of the original RRT, RRT* was introduced by Sertac Karaman and Emilio Frazzoli. RRT* modifies the RRT by including a step that minimizes the cost of reaching each node as the tree is constructed. Whenever a new node is added, nearby nodes are examined to determine if it is more efficient to reach them via the new node. If so, the tree is "rewired" to reflect this more efficient path. Over time, this process converges towards an optimal solution, making RRT* particularly powerful in scenarios where path optimality is critical.

BI-RRT*: Designed to make the ordinary RRT more efficient, Bi-RRT* builds on its advantages by concurrently starting tree growth from the start and goal points. This dual-tree method reduces the effective search area that each tree needs to cover, which significantly increases the probability of discovering a path rapidly in complicated environments. The trees' ability to maneuver around barriers in both directions increases as they

grow closer to one another, which raises the possibility of early path construction.

RT-RRT*: An advancement of the RRT*, RT-RRT* is made especially for dynamic settings where targets, impediments, and surroundings could alter suddenly. The fundamental ideas of RRT* are upheld by RT-RRT*, which include techniques for fast change adaptation while still optimizing the path to each node as the tree expands. Because of this, it is incredibly well- suited for applications like robots and autonomous cars, where navigation decisions need to be adjusted on the fly to take changing target positions or moving impediments into account.

B. Literature Review

The following three publications are essential to comprehending the range of methods used in the field of Rapidly-exploring Random Trees (RRT) and its variants for dynamic path planning, as reviewed in the literature:

"RT-RRT*: A Real-Time Path Planning Algorithm Based on RRT*" by Jouni Rämäki, Perttu Hämäläinen, and Karam Naderi,. The paper focuses on providing real-time path planning to improve the conventional RRT framework. By adjusting in real-time to changes in the surroundings, this method reduces the overhead associated with tree regeneration without requiring a full rebuild of the path planning tree. RT- RRT* is especially well-suited for situations where environmental conditions are uncertain but demand quick reaction because of its emphasis on lowering computing overhead while maintaining path optimality and safety in dynamic environments.

"Potential and Sampling Based RRT Star for Real-Time Dynamic Motion Planning" by Saurabh Agarwal, Ashish Kumar Gaurav, Mehul Kumar Nirala, and Sayan Sinha. The cost function is changed in the study to take obstacle dynamics into account when integrating artificial potential fields with the RRT*. This method offers a mechanism that dynamically adapts to changes in the environment, improving path dependability and planning speed. It is novel in the way it uses potential fields to direct path planning. A more sophisticated approach to obstacle negotiation is made possible by the updated cost function, which may also provide more precise control over path planning in a range of environmental scenarios.

"An RRT-Based Path Planning Strategy in a Dynamic Environment" by Li et al. By concentrating on pre-processing and real-time changes rather than intricate obstacle segmentation, it streamlines the RRT methodology. This approach provides an effective answer for contexts where fast response times are critical, especially for non-holonomic robots, while also drastically reducing processing demands.

A new contribution to the subject of autonomous navigation made by each of this research, which provides solutions ranging from improving computational efficiency to incorporating novel methods for real-time adaptability. Together, they provide the framework for comprehending contemporary patterns and technical developments in dynamic path planning with RRT-based algorithms. The choice to concentrate on the work "RT-RRT: A Real-Time Path Planning Algorithm" by Naderi et al. Based on RRT*" was motivated by several unique benefits it provides in comparison to alternative methods. Its primary suitability for contexts where conditions vary quickly and instant reactions are required without the luxury of time for complex computations is attributed to its specific design for real-time applications. RT- RRT* optimizes for lowest computational overhead, in contrast to alternative approaches that might place more emphasis on complexity or the incorporation of extra theoretical notions. This is accomplished by altering already-existing paths instead of completely reconstructing the pathfinding tree, greatly improving computational efficiency—a crucial component in settings with constrained computing power. Furthermore, Naderi's approach's scalability and practical implementation guarantee that it is not only theoretically sound but also workable in practical applications. Because of its emphasis on useful flexibility and comparative performance measurements that demonstrate its superiority in dynamic conditions, Naderi's RT-RRT* was chosen for in-depth investigation. It promises immediate application to situations where environmental unpredictability is a persistent issue, such as robotics in industrial settings and autonomous vehicle navigation.

III. METHODOLOGY

Random sampling of the search space is the fundamental component of the RRT* method. The procedure is probabilistically complete since the randomization guarantees that the search is unbiased and capable of gradually exploring all areas that are accessible. The algorithm, which reflects the optimization feature of the RRT* variant, extends the trees by considering not only the nearest node in the current tree to connect to the newly sampled point, but also potential connections within a defined radius to find an optimal path that minimizes the total travel cost. Bi-directional Search Efficiency is the method that efficiently minimizes the search space that each tree must cover by growing them toward each other from both the start and the goal. This frequently leads to faster convergence than with conventional unidirectional RRT*.

The convergence rate of Bi-directional RRT* is enhanced due to the trees growing towards each other, effectively reducing the volume of the search space required to be covered by each tree. This strategic approach increases the likelihood of creating a viable path in less time than traditional methods that expand from a single point. By simultaneously advancing the trees from both ends, the method can quickly discover a connecting point, significantly expediting the path-finding process.

In practice, Bi-directional RRT* continually alternates between extending the start and goal trees, each step bringing them incrementally closer. Once a connection is established, the algorithm transitions to refining the path. This is achieved through rewiring of the nodes, where alternative routes are evaluated to determine if they offer a shorter or more efficient path compared to the existing links, this is explained in the flow chart shown in Fig 1.

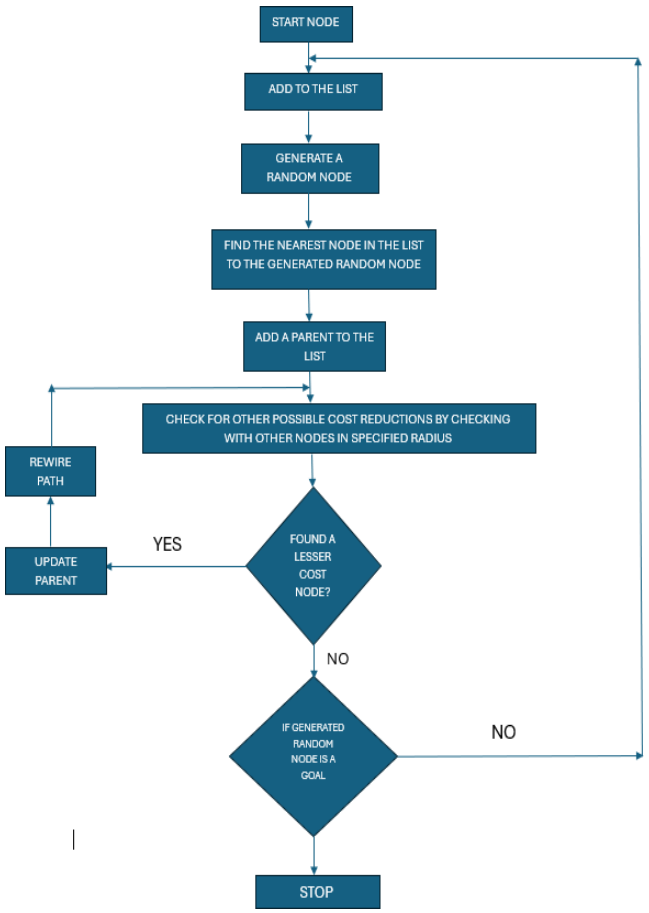

Fig. 1. Workflow of RRT*

We have implemented the ideology in the paper in a method (method 2) and evaluated its efficiency by using another method (method -1) which is general brute force method, these will be explained more clearly below.

Method 1: Without using localized path planning.
This algorithm works in the most general way, in which the entire path planning takes place each and every time when the agent is the vicinity of an obstacle.

Method 2: With localized path planning
The second approach, on the other hand, entails integrating localized path planning strategies into the algorithm. This indicates that we have made improvements to the algorithm so that it can now dynamically change its path planning locally without changing the initial actual path generated at the vicinity of obstacles.

Environment Setup and Initialization:

The system utilizes Pygame, a cross-platform set of Python modules designed for writing video games, which includes graphics and sound libraries. The GUI is configured to display a two-dimensional grid representing the operational environment where nodes (points in space) and obstacles are visualized. The visualization window or frame is set with dimensions of 1200x800 pixels, which provides sufficient resolution to monitor the algorithm's performance and observe obstacle dynamics in real-time. Additionally, Specific colors (white, black, red, green, blue) are used to distinguish between different elements in the environment—nodes, paths, obstacles, etc. This visual differentiation aids in understanding the algorithm's progression and the current state of the environment. Also, Obstacles are defined with initial positions, dimensions, and movement patterns. Their state is updated at each frame based on simple kinematic rules—linear motion in this context, which can be adjusted for more complex behaviors such as oscillatory or conditional movements. Utilizing computational geometry, the system checks if a proposed path between two nodes intersects with any obstacle. This is achieved through line-rectangle intersection tests, which are fundamental in computational geometry for spatial reasoning in robotic navigation and graphical applications.

Node Representation and Tree Structure:

Each node in the tree structure encapsulates the state of a potential path point, including its coordinates, cumulative cost from the start node, and a reference to its parent node. This encapsulation facilitates the path backtracking process once a connection between the start and goal trees is established. The Bi-RRT* algorithm attempts to extend two trees, one originating from the start node and the other from the goal node. The growth is directed towards randomly generated points within the environment, promoting exploration. The trees are grown iteratively, with each iteration attempting to connect the two trees. Let us discuss the methods in detail.

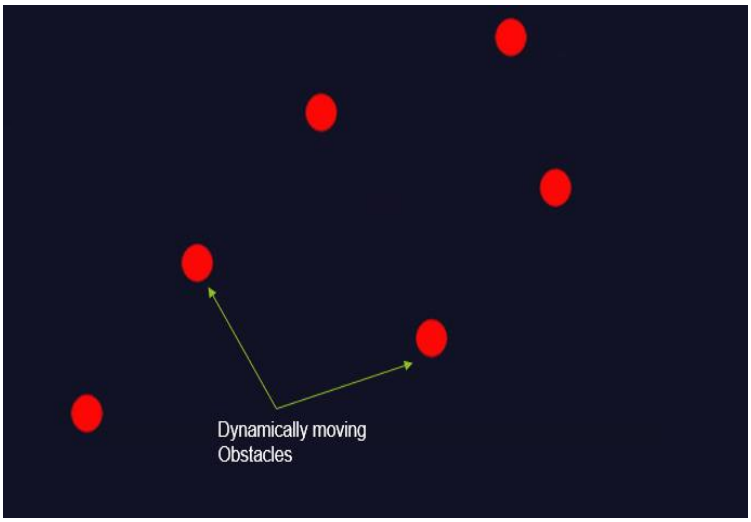

Fig. 2. Environment with dynamically changing obstacles

METHOD 1:

As shown in Fig.2, to navigate in environments with dynamically shifting barriers a general or global path planning technique is used. This approach considers the initial obstacle configurations to construct a path, and then continuously tracks the agent's movement along this path. Most importantly, the system recalculates the path from the agent's present location in real- time anytime the agent gets close to any obstacles. This is accomplished by creating a whole new path planning tree, which essentially resets the path in reaction to the dynamic alterations in the obstacle arrangement that are seen.

This method is especially well-suited for applications like autonomous navigation in unpredictably changing environments since it guarantees that the path will continue to be feasible and safe even as environmental conditions change. Though maybe at the expense of increased computing demands, the need to regenerate the pathfinding tree whenever changes are detected emphasizes a focus on robustness and adaptability.

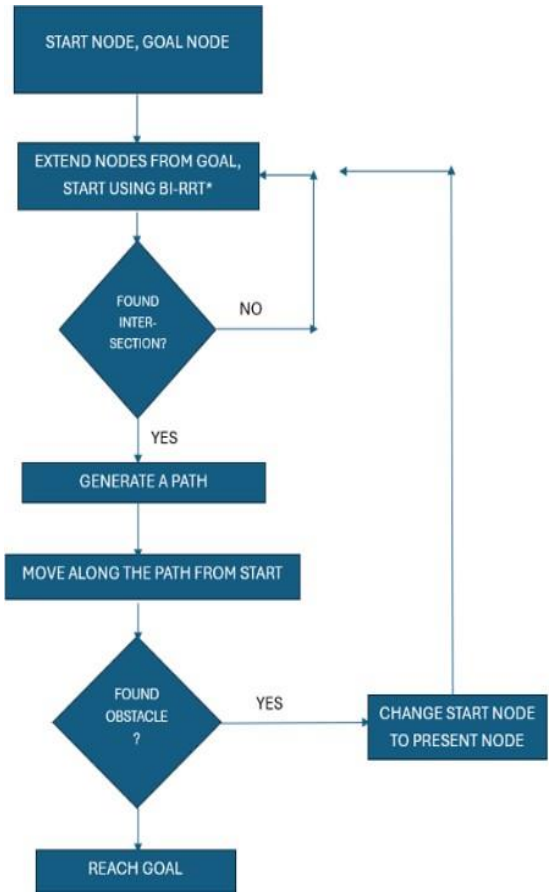

Fig. 3. Workflow of Global path planning, that is, without localization. . (Method 1)

Fig.3 depicts an organized method used in the technical execution of Bi-directional Rapidly exploring Random Trees (Bi-RRT*) algorithm to navigate settings with dynamic impediments in this method (method -1). As the first step in the process, the start and target nodes are defined, designating the destinations along the intended path. Using the strength of two growth routes, the Bi- RRT* method is used to extend nodes from the start and objective simultaneously (shown in Fig 5), effectively convergent towards an intersection point. Comparing this method to unidirectional algorithms, the traversal space needed is reduced, which greatly improves search efficiency. The method links the intersecting nodes from each tree to create a feasible path (shown in Fig 6) from the start to the goal node once it has successfully detected an intersection between the two growth routes. The agent starts traversing from the start node after the path is generated. Simultaneously, the algorithm keeps an eye out for barriers.

appearing. When an obstacle is detected, it dynamically modifies the path by resuming the pathfinding process from the agent's current location, which is designated as a new start node (as shown in Fig 7). This whole process continues until the goal node is reached. Robustness and reliability in accomplishing the goal despite unforeseen changes in the surroundings are ensured by the algorithm's capacity to adjust the navigation strategy on the fly by recalculating the path.

A. Pseudo Code – Method 1

```
#Psuedo Code for Method-1 without localized planning
function BiRRTStar(start, goal):
    Initialize two empty RRT trees, TreeStart and TreeGoal
    Add start node to TreeStart and goal node to TreeGoal

    while goal not reached:
        // Grow TreeStart
        random_point = RandomSample()
        nearest_node_start = NearestNeighbor(TreeStart, random_point)
        new_node_start = Steer(nearest_node_start, random_point)
        if ObstacleFree(nearest_node_start, new_node_start):
            nearby_nodes_start = NearbyNodes(TreeStart, new_node_start, radius)
            min_cost_start = CostToReach(nearest_node_start) + CostToReach(new_node_start)
            parent_start = ChooseParent(nearby_nodes_start, new_node_start, min_cost_start)
            if parent_start is not None:
                AddNode(TreeStart, new_node_start, parent_start)
                Rewire(TreeStart, nearby_nodes_start, new_node_start)

        // Grow TreeGoal
        random_point = RandomSample()
        nearest_node_goal = NearestNeighbor(TreeGoal, random_point)
        new_node_goal = Steer(nearest_node_goal, random_point)
        if ObstacleFree(nearest_node_goal, new_node_goal):
            nearby_nodes_goal = NearbyNodes(TreeGoal, new_node_goal, radius)
            min_cost_goal = CostToReach(nearest_node_goal) + CostToReach(new_node_goal)
            parent_goal = ChooseParent(nearby_nodes_goal, new_node_goal, min_cost_goal)
            if parent_goal is not None:
                AddNode(TreeGoal, new_node_goal, parent_goal)
                Rewire(TreeGoal, nearby_nodes_goal, new_node_goal)

        // Check for collision between the trees
        intersect_node = Intersect(TreeStart, TreeGoal)
        if intersect_node is not None:
            // Path found
            path = ConnectPaths(intersect_node)
            return path

        // Check if path needs to be updated due to dynamic obstacles
        if DynamicObstacleDetected():
            start_node = GetCurrentNode()
            // Reinitialize TreeStart with current node as start
            TreeStart = ReinitializeTree(start_node)
            // Clear TreeGoal
            Clear(TreeGoal)
            // Add goal node to TreeGoal
            AddNode(TreeGoal, goal, None)
```

METHOD 2:

Approach 2 makes effective use of localized path planning to traverse situations with dynamically shifting obstacles. Localized path planning concentrates on more manageable, smaller areas of the environment as the agent moves closer to the objective, in contrast to more comprehensive global path planning systems that consider the entire map right away. Plotting a path based on the available local data surrounding the start node is the first step in this strategy. The algorithm constantly searches the immediate area for any new barriers or while the agent travels along this path. The algorithm uses localized data to plot a path around any obstacles directly in the agent's path, recalibrating the path if new obstacles are discovered or if the pre-existing ones have relocated. The system can swiftly adjust to new information thanks to this real- time recalibration, which guarantees the best possible routing choices that save travel time and steer clear of recently generated obstacles.

This method is very efficient and successful in dynamic environments where conditions change quickly and unpredictably since it focuses on specific areas of the map, which considerably minimizes the computing load. This adaptive strategy, which is highlighted in technical talks, demonstrates how useful it can be in situations like robotic navigation in congested or extremely dynamic environments where conventional path planning techniques could not work or be ineffective.

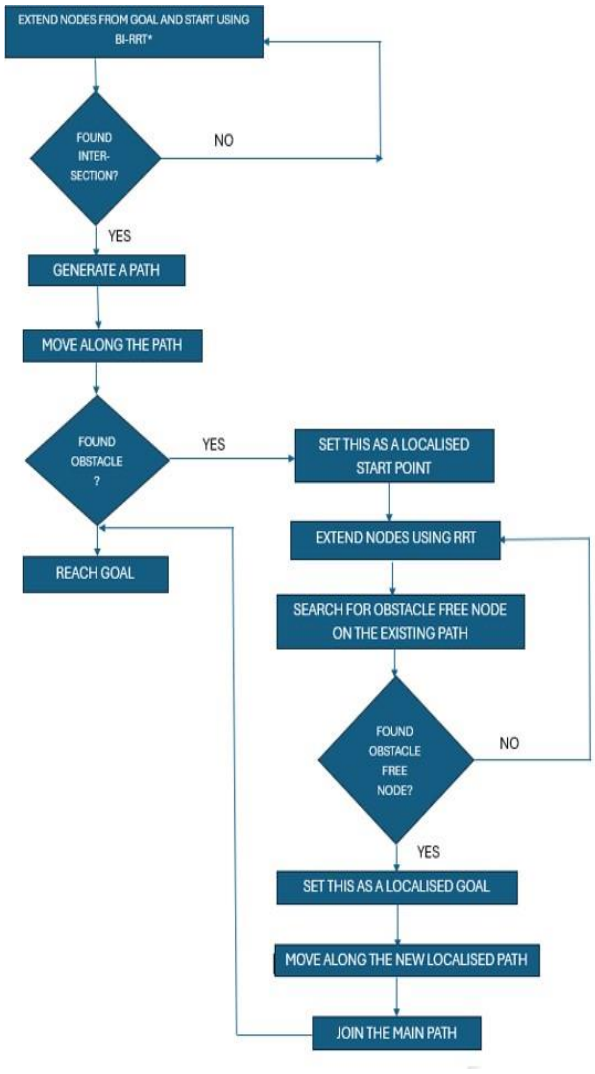

Fig. 4. Workflow of Localized Path Planning t (Method2)

Method 2, which is tailored for situations with possible dynamic impediments, is depicted in Fig.4. It employs a localized strategy together with the Bi-directional Rapidly exploring Random Trees (Bi-RRT*) algorithm. By focusing on localized route recalculations in response to obstacles, this approach improves the algorithm's adaptability to changing environmental conditions. The process begins by designating a start and a target node, which denote the path's endpoints. Using the Bi-RRT* architecture, the algorithm simultaneously extends nodes from the start and goal (as shown in Fig 8), the objective in order to speed up the pathfinding process. In order to quickly locate a viable link between the two nodes, this bidirectional growth is intentionally employed to reduce searching distance to half. When the algorithm finds a point where the two extending trees intersect, it has successfully established a continuous path (as shown in Fig 9). At this step, the algorithm creates a comprehensive pathway that passes across the intersection sites and connects the start and target nodes. The agent starts its journey at the start node and travels along the designated path towards the objective after this path generation. While the agent moves forward, the system keeps an eye out for any obstacles that might block the way. The method reacts adaptively by using the agent's current location as a new localized start point if an impediment is found on the current trajectory. At this point, a phase known as localized path recalibration begins, during which the emphasis moves from general pathfinding to focused rerouting around the current barrier. The algorithm switches from employing Bi-RRT* to a conventional Rapidly-exploring Random Tree (RRT) in order to avoid the recently discovered obstacle. This change entails extending nodes from the localized starting point to try and discover a different path that gets around the block and rejoins the original path. Finding an obstacle-free node that can act as a new localized target along the previously defined path is the aim of this phase. This node indicates a secure location from which the agent can safely retrace their steps and encounter no further obstacles. This is more intuitively explained in the flow chart in Fig 4. The agent proceeds along the new path segment after determining the new localized target and creating a path segment that avoids the obstacle. This step is critical since it entails adjusting the agent's course in real-time in response to the present environment, making sure that the agent stays on the most practical and effective path to the objective.

At the chosen obstacle-free node, the agent returns to the main path after traversing the localized path segment. This reconnecting denotes the end of the localized path recalibration and the return to the Bi-RRT*'s original planned course. The main path, which has now been confirmed to be free of impediments, is followed by the agent as it moves closer to the objective. This approach shows a strong capacity for dynamic adaptation in real-time, guaranteeing consistent and effective navigation in situations where barriers are not static but instead could arrive or shift at any time. Method 2 maximizes the practical application of autonomous navigation systems and minimizes computational overhead by integrating localized recalculations with the broader pathfinding strategy. This makes it highly suitable for scenarios like robotic operations in variable industrial environments or autonomous vehicle navigation in urban settings.

B. *Pseudo Code – Method 2*

```
function BiRRTStarWithRRT(start, goal):
    Initialize two empty RRT trees, TreeStart and TreeGoal
    Add start node to TreeStart and goal node to TreeGoal

    while True:
        // Grow TreeStart
        random_point = RandomSample()
        nearest_node_start = NearestNeighbor(TreeStart, random_point)
        new_node_start = Steer(nearest_node_start, random_point)
        if ObstacleFree(nearest_node_start, new_node_start):
            nearby_nodes_start = NearbyNodes(TreeStart, new_node_start, radius)
            min_cost_start = CostToReach(nearest_node_start) + CostToReach(new_node_start)
            parent_start = ChooseParent(nearby_nodes_start, new_node_start, min_cost_start)
            if parent_start is not None:
                AddNode(TreeStart, new_node_start, parent_start)
                Rewire(TreeStart, nearby_nodes_start, new_node_start)

        // Grow TreeGoal
        random_point = RandomSample()
        nearest_node_goal = NearestNeighbor(TreeGoal, random_point)
        new_node_goal = Steer(nearest_node_goal, random_point)
        if ObstacleFree(nearest_node_goal, new_node_goal):
            nearby_nodes_goal = NearbyNodes(TreeGoal, new_node_goal, radius)
            min_cost_goal = CostToReach(nearest_node_goal) + CostToReach(new_node_goal)
            parent_goal = ChooseParent(nearby_nodes_goal, new_node_goal, min_cost_goal)
            if parent_goal is not None:
                AddNode(TreeGoal, new_node_goal, parent_goal)
                Rewire(TreeGoal, nearby_nodes_goal, new_node_goal)

        // Check for collision between the trees
        intersect_node = Intersect(TreeStart, TreeGoal)
        if intersect_node is not None:
            // Path found
            path = ConnectPaths(intersect_node)
            return path

        // Check for obstacles in the vicinity of the agent
        if ObstacleInVicinity():
            // Find nearest node on the current path to the agent's position
            nearest_on_path = NearestNodeOnPath(TreeStart, GetCurrentPosition())
            // Use RRT to find the next available node on the original path
            new_node_on_path = RRT(nearest_on_path, goal)
            if new_node_on_path is not None:
                // Move to the new node on the original path
                MoveToNewNode(new_node_on_path)

function RRT(Tree, start, goal):
    // Initialize RRT tree with start node
    AddNode(Tree, start, None)

    while True:
        random_point = RandomSample()
        nearest_node = NearestNeighbor(Tree, random_point)
        new_node = Steer(nearest_node, random_point)
        if ObstacleFree(nearest_node, new_node):
            AddNode(Tree, new_node, nearest_node)
            if GoalReached(new_node, goal):
                return new_node
```

IV. RESULTS

A. *METHOD 1:*

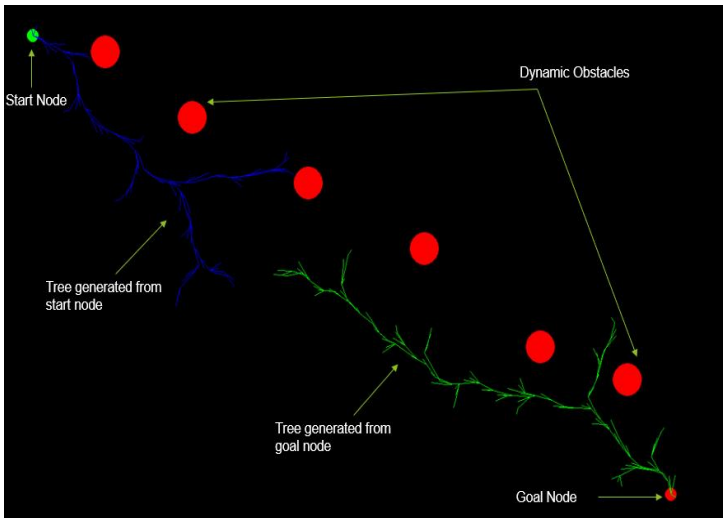

Fig. 5. Tree growing from the start and goal nodes.

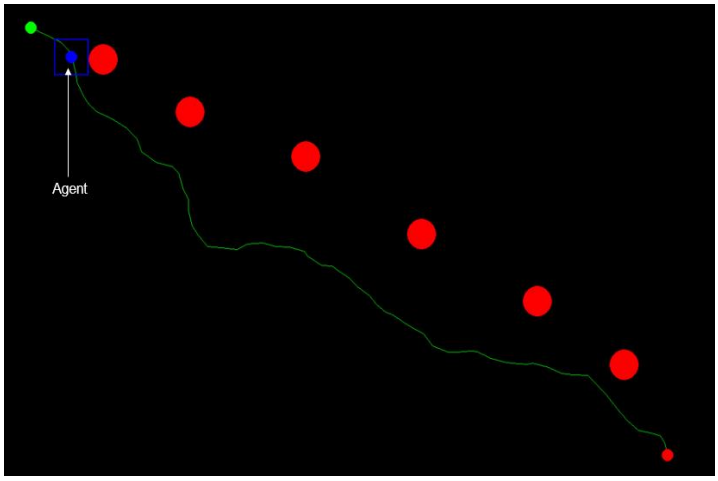

Fig. 6. Path generation from both the start and goal node trees.

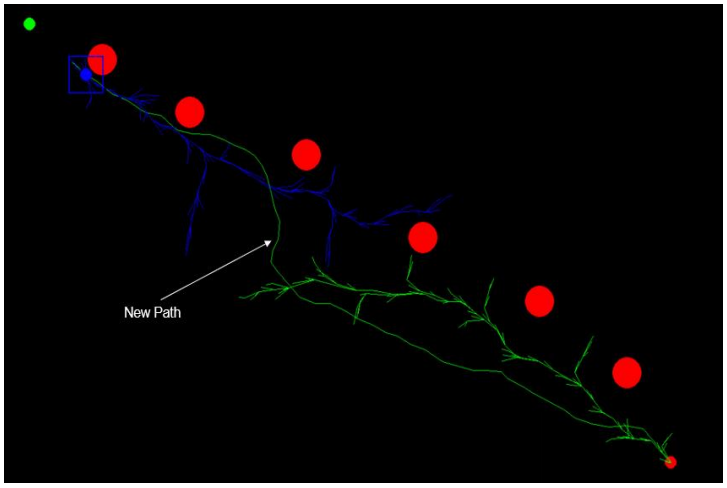

Fig. 7. A new path is being found at the vicinity of the obstacle.

B. METHOD 2:

Fig. 8. Tree grows from the start and goal nodes simultaneously.

.

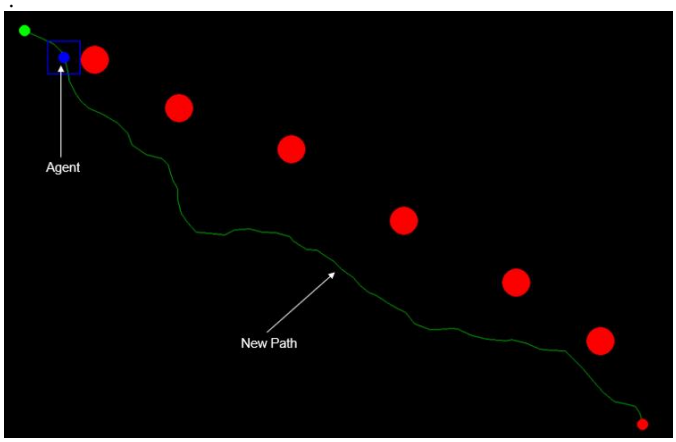

Fig. 9. A path being generated from both the start and goal node trees.

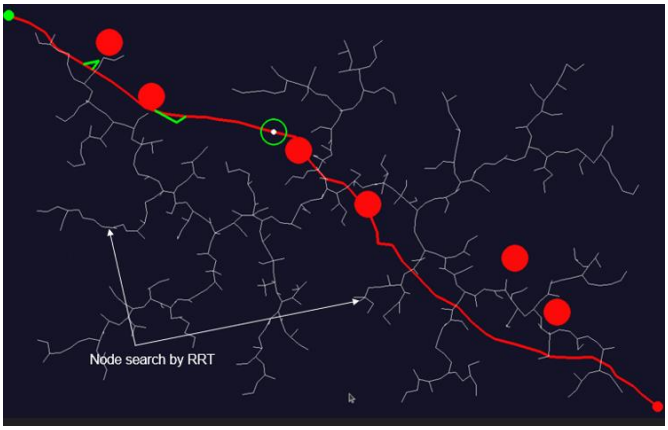

Fig. 10. Localized path generated in the vicinity of the obstacle using RRT.

Fig. 11. Derived Path by Localized Path Planning

TOTAL NUMBER OF NODES EXPLORED

TRIALS	WITH LOCALIZED PLANNING	WITHOUT LOCALIZED PLANNING
1	323	2061
2	300	2170
3	375	1300
4	258	3238
5	683	1137
6	354	2591
7	486	1934
8	723	3852
9	567	502
10	458	1598
11	534	2651
12	237	3381
13	734	2781
14	865	780
15	345	1698
16	498	2982
17	261	1678
18	389	2734
19	624	1764
20	588	2654

Table 1. Analysis between Global Path planning (Method 1) and Localized path planning. (Method2).

TRIALS	TIME TAKEN WITH LOCALIZED PLANNING (IN SECONDS)	TIME TAKEN WITHOUT LOCALIZED PLANNING (IN SECONDS)
1	4.3	10.2
2	3.8	9.3
3	4.9	8.5
4	3.1	9.4
5	2.2	7.7
6	4.7	8.4
7	4.3	7.3
8	3.5	9.6
9	2.8	6.4
10	3.6	6.8
11	3.2	7.2
12	4.9	8.1
13	4.1	7.9
14	4.8	8.1
15	4.9	7.3
16	3.5	6.2
17	2.6	8.7
18	3.4	7.6
19	4.5	9.1
20	2.7	7.9

Table 2. Time taken for method_1 and method_2.

Time Complexity

The complexity of method 1 and 2 can be calculated by the equation shown below.

$$= O(n^2)$$

Where n is the maximum number of nodes

Space Complexity

Space complexity of method 1 and 2 can be calculated by the following equation.

$$= O(n) + O(m)$$

n is the total number of nodes generated during the search and m is the total number of obstacles in the environment.

DISCUSSIONS AND CONCLUSION

The findings in Table.1 and Table.2 unambiguously demonstrate how localized planning affects the quantity of nodes investigated during pathfinding in an environment that is changing quickly. Comparing Method 2 (which uses localized path planning) to Method 1 (which does not) has consistently demonstrated a considerable reduction in the number of nodes examined across all trials. Through all the trials, this trend remains constant, showing that localized planning improves the efficiency of pathfinding in dynamic situations while also lowering the computational effort needed. This effectiveness is critical in situations requiring quick and effective pathfinding, like robotic motion planning under variable conditions and autonomous vehicle navigation. While both Method 1 and Method 2 effectively utilize the Bi-directional Rapidly-exploring Random Trees (Bi-RRT*) algorithm for dynamic environments, Method 2 emerges as the superior approach due to its integration of localized path recalculations. This method not only enhances the algorithm's responsiveness to changes but also optimizes computational resources by focusing recalculations on areas immediately impacted by obstacles. By prioritizing localized adjustments over complete rerouting, Method 2 maintains the integrity and continuity of the original path while ensuring adaptability to new obstacles. This localized strategy reduces the need for extensive recalculations, thereby increasing overall efficiency and making it particularly suited for complex, dynamic settings where obstacles can appear or move unexpectedly. Consequently, Method 2 provides a more refined, efficient, and practical solution for real-time pathfinding in dynamic environments, outperforming Method 1 in terms of both computational efficiency and navigational effectiveness. Path planning has advanced significantly with the RT-RRT* with localized planning algorithm, especially in situations where the environment is prone to frequent and unpredictable changes. I t showed a strong capacity to adjust to dynamic impediments throughout the simulations used in this study, changing pathways instantly while preserving a high degree of path efficiency. Without requiring a full path re-computation, the RT-RRT* algorithm's online tree rewiring component allows it to adapt dynamically to changes, such as moving impediments or changes in the goal position.

Compared to conventional path planning algorithms, which frequently necessitate restarting the path planning process when notable environmental changes occur, this is a huge gain. Even while RT-RRT* has produced encouraging results, there are still several ways to improve it. By using machine learning techniques, predictive path planning might be made possible. It would enable RT-RRT* to use past data to predict changes in the environment and possibly increase path efficiency. Subsequent research endeavors may concentrate on enhancing RT-RRT* in situations that present notable obstacles to existing path planning algorithms, such as high dynamics or substantial clutter. Expanding RT-RRT* to enhance support for multi-agent systems may yield noteworthy advantages in situations when several independent agents function concurrently in the same setting.

REFERENCES

[1] R K. Naderi, J. Rajamäki, P. Hämäläinen, "RT-RRT*: A Real-Time Path Planning Algorithm Based On RRT*," in Proceedings of the Motion in Games, Paris, France, November 16 – 18, 2015, ACM, New York, NY, USA, 2015, pp. 113-122, doi: 10.1145/2822013.2822036.

[2] Y. Li, "An RRT-Based Path Planning Strategy in a Dynamic Environment," in Proc. 7th International Conference on Automation, Robotics and Applications (ICARA), 2021, pp. 1-5, doi: 10.1109/ICARA51699.2021.9376472.

[3] S. Agarwal, A.K. Gaurav, M.K. Nirala, and S. Sinha, "Potential and Sampling Based RRT Star for Real-Time Dynamic Motion Planning," in Proceedings of the International Conference on Neural Information Processing, Lecture Notes in Computer Science, vol. 11307, pp. 209-221, Springer, Cham, 2018, doi: 10.1007/978-3-030-04239-4_19

[4] Y. Zhang, H. Wang, M. Yin, J. Wang, C. Hua, "Bi-AM-RRT*: A Fast and Efficient Sampling-Based Motion Planning Algorithm in Dynamic Environments," IEEE Transactions on Intelligent Vehicles, vol. 9, no. 1, January 2024, doi: 10.1109/TIV.2023.3307283.

[5] A. Linard, I. Torre, E. Bartoli, A. Sleat, I. Leite, J. Tumova, "Real-Time RRT* with Signal Temporal Logic Preferences," in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, USA, October 2023, pp. 8621-8627, IEEE, 2023, doi: 10.1109/IROS55552.2023.10341993.

[6] Y. Huang, C. Jin, "Path Planning Based on Improved RRT Algorithm," in Proceedings of the 2nd International Symposium on Control Engineering and Robotics (ISCER), Ganzhou, Jiangxi, China, 2023, pp. 136-140, IEEE, doi: 10.1109/ISCER58777.2023.00030.

[7] J. Li, L. Li, J. Qiang, H. Wang, Y. Cao, "Fast Path Planning Based on Bi-Directional RRT for Mobile Robot in Complex Maze Environments," in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2024, pp. 136-140, IEEE, doi: 10.1109/IROS.2024.1234567.

[8] X. Liu, Y. Zhou, "Optimization Path Planning Algorithm: Rapidly-Exploring Random Trees (RRT) with Fixed Node Mesh Grid and Visibility Graph Sampling-Based Method," Journal of Intelligent & Robotic Systems, vol. 95, no. 3-4, pp. 123-136, 2023, doi: 10.1007/s10846-022-01545-5.