2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Real-Time Fast Marching Tree for Mobile Robot Motion Planning in Dynamic Environments*

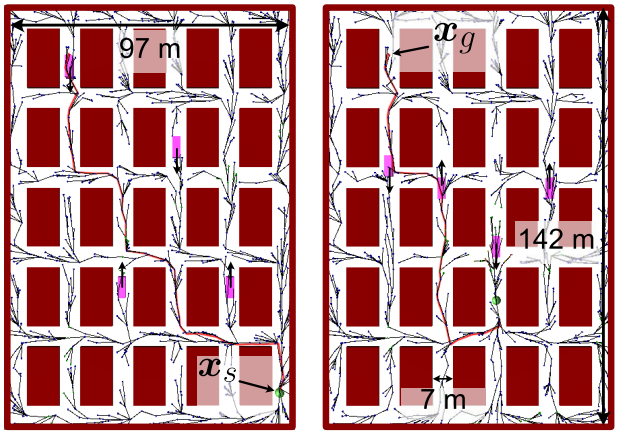
Jefferson Silveira[1], Kleber Cabral[2], Sidney Givigi[3] and Joshua A. Marshall[4]

***Abstract*— This paper proposes the Real-Time Fast Marching Tree (RT-FMT), a real-time planning algorithm that features local and global path generation, multiple-query planning, and dynamic obstacle avoidance. During the search, RT-FMT quickly looks for the global solution and, in the meantime, generates local paths that can be used by the robot to start execution faster. In addition, our algorithm constantly rewires the tree to keep branches from forming inside the dynamic obstacles and to maintain the tree root near the robot, which allows the tree to be reused multiple times for different goals. Our algorithm is based on the planners Fast Marching Tree (FMT*) and Real-time Rapidly-Exploring Random Tree (RT-RRT*). We show via simulations that RT-FMT outperforms RT-RRT* in both execution cost and arrival time, in most cases. Moreover, we also demonstrate via simulation that it is worthwhile taking the local path before the global path is available in order to reduce arrival time, even though there is a small possibility of taking an inferior path.**

I. INTRODUCTION

Planning optimal paths for mobile robots in environments filled with obstacles is a complex task that may require considerable computation time. This can cause issues in time-sensitive applications, such as mining [1] because the longer it takes to execute a task, the lower the efficiency and the higher the expenses. This is also true for search and rescue applications [2] since any saved time is used to look for survivors. These are applications where it is possible for a robot to encounter dynamic obstacles such as humans or other robots. To deal with such scenarios, this paper proposes a new path planning algorithm that works in real-time, thus reducing unnecessary waiting time while still avoiding dynamic obstacles.

In dynamic environments, new paths must be generated whenever a path becomes unfeasible. It is possible to save planning time by incorporating a re-planning approach that leverages the current search structure (e.g., a graph) that has been created previously instead of starting a new instance from scratch. One common approach to reducing planning time for time-sensitive applications is to generate a sub-optimal but complete path and start the path execution as soon as possible while it continues to optimize the path. Algorithms that fall into this category are, for example, the anytime planners [3], [4]. A different approach that can be applied when the application has hard time constraints is to expand the search until it is time to perform an action. In this case, actions are taken before the planner finds the complete trajectory. Algorithms that fall into this category are called *real-time planners* [5].



(a) (b)

Fig. 1: Illustration of a mine-like space. The start and goal configurations are shown as x_s and x_g, respectively. The solution is highlighted in red. The pink squares are the dynamic obstacles representing mining trucks and the dark solid obstacles are the mine pillars. The goal is to traverse the space with the least amount of time to maximize profit.

While real-time planners provide the fastest trajectory execution, which may allow the robot to arrive at the destination sooner, there are two problems associated with these techniques. At the time to perform an action, the planner applies a heuristic function to decide the best local path to follow with the currently available information. Local minima in the heuristic function can cause the robot to visit unnecessary states. In addition, the planner may lead a robot to a state of unavoidable collision if there is not enough look-ahead [6].

These problems can be greatly reduced if the planner is able to quickly search through the space to provide sufficient look-ahead for the real-time planner to select low-

*This research was supported in part by the Natural Science and Engineering Research Council of Canada (NSERC) under grant number DNDPJ 533392-18, General Dynamics Land Systems (Canada), and Defence R&D Canada (DRDC).

[1] J. Silveira is with the Department of Electrical & Computer Engineering and the Ingenuity Labs Research Institute, Queen's University, Kingston, ON K7L 3N6 Canada jefferson.silveira@queensu.ca

[2]K. Cabral is with the School of Computing, Queen's University, Kingston, ON K7L 3N6 Canada kleber.cabral@queensu.ca

[3]S. Givigi is with the School of Computing and the Ingenuity Labs Research Institute, Queen's University, Kingston, ON K7L 3N6 Canada sidney.givigi@queensu.ca

[4]J. Marshall is with the Department of Electrical & Computer Engineering and the Ingenuity Labs Research Institute, Queen's University, Kingston, ON K7L 3N6 Canada joshua.marshall@queensu.ca

cost local paths while it searches for the optimal global path. Fast search is a characteristic of probabilistic sampling-based planners such as RRT* [7] and FMT* [8], which have been successfully applied in robotic problems for high-dimensional configuration spaces due to the speed at which these algorithms are able to search over the configuration space [7]. These types of planners are clearly good candidates for real-time applications. To fill this gap, the real-time variant RT-RRT* was proposed by Naderi et al. [5]. This variant is capable of leveraging the generated tree to avoid dynamic obstacles by rewiring the tree around them as the robot executes the generated path. In addition, it generates local paths for execution before the complete path is found for real-time execution.

Unfortunately, RRT variants have a common drawback. In environments with many obstacles, there is a low probability that a sample will be added to the tree due to collisions, requiring more iterations for the planner to find a solution. In contrast, the FMT* algorithm starts the search with all samples already distributed in the environment. As a result, FMT* provides solutions in less time and with lower costs than RRT*. This effect is discussed in more detail in [8].

Inspired by the advantages of FMT* over RRT*, this paper proposes RT-FMT[1], a real-time variant of the FMT* that features local path generation, multiple-query planning, and dynamic obstacles avoidance. During the search, our approach quickly looks for the global solution and, in the meantime, generates local paths that can be used by the robot to start execution faster. In addition, our algorithm constantly updates the tree to avoid dynamic obstacles, and to maintain the tree root near the robot, which allows the tree to be reused multiple times for different goals. Fig. 1 shows the RT-FMT in a layout similar to an idealized underground mine. Fig. 1a shows the initial stage with the solution highlighted in red, and Fig. 1b shows the environment some time later. We can see that the tree root is near the robot in both images and that the path has been updated to avoid the nearest truck (pink rectangle) in Fig. 1b.

The performance of RT-FMT was evaluated against RT-RRT* in two dynamic environments. The results (see Section V) show that robots applying RT-FMT consistently arrived at the goal state before RT-RRT*. In addition, and they also show that when the robots start an early execution, risking moving towards sub-optimal paths, they consistently arrived at the goal state faster when compared to waiting for the complete optimal path.

The remainder of this paper is structured as follows. Section II presents some related works on planning in dynamic environments with online execution. Section III presents the description of the proposed algorithm. Section IV presents the evaluation methodology. Section V presents and discusses the results of the simulated experiments, and Section VI presents some final remarks and future work.

[1]We decided to drop the "*" in FMT* when naming our approach since real-time execution on local paths does not guarantee optimality

II. RELATED WORK

In this section, we review the literature on online and real-time sampling-based planners for dynamic environments. Although there are algorithms that plan with prior information on the trajectories of the obstacles such as [9], we focus on the techniques that only require the current position of the dynamic obstacles because it is easier to obtain such information from sensors or communication.

The CL-RRT [10] is an approach that integrates an online controller with RRT in order to plan in the presence of uncertainty and still satisfy bounds on tracking error. When the robot moves along the generated path, the algorithm prunes infeasible branches. However, it does not update the root of the tree as RT-RRT* does. In addition, it has been shown in [5], that RT-RRT* is able to find shorter paths with fewer iterations in simpler systems without motion constraints. Online versions of the RRT* (ORRT*) and FMT* (OFMT*) have been proposed by Chandler and Goodrich [11]. Their approach is similar to RT-RRT* in the sense that multiple goals can be assigned without having to re-plan from scratch, but they apply a different rewiring approach. While RT-RRT* rewires around obstacles by checking for collisions, ORRT* and OFMT* apply a time-varying cost that increases near obstacles, causing nodes in the tree to prefer connections far from the obstacles. Another difference between these algorithms is the local path generation process that is part of RT-RRT*, which is not present in ORRT* and OFMT*. More recently, Tong et al. [12] proposed RRT*FN-Replan, which is an online algorithm that maintains a fixed number of nodes in the tree and that takes advantage of previously generated tree branches to update the current plan. They have shown that their approach outperforms not only RT-RRT* but also ORRT*.

Based on the current state of the art, we decided to implement RT-FMT and compare it against RT-RRT* for a few reasons. First, it has been shown to outperform CL-RRT. ORRT* and OFMT* do not include local path generation, and the obstacle avoidance module (based on cost) guarantees safe execution at the expense of higher cost trajectories. Even though it has been shown that RRT*FN-Replan outperforms RT-RRT*, these experiments involved testing only the rewire and re-plan processes, disregarding the first stage of tree expansion. Because the arrival time is directly affected as well by the tree expansion, it was deemed important to also consider this information.

III. THE REAL-TIME FAST MARCHING TREE ALGORITHM

In general, RT-FMT works by expanding a tree, similarly to how FMT* does, while it also checks for dynamic obstacles and rewires the tree around them. During the search, the algorithm also searches for the local paths with the least costs. These paths are used for the robot to start moving before planning is finished. When the robot reaches a new waypoint in the path, the root of the tree is updated. This event triggers a complete rewire of the tree to update the costs of all nodes. In a real-time application, the robot sends

Algorithm 1: RT-FMT($\boldsymbol{x}_s, \boldsymbol{x}_g, \mathcal{X}_{Fobs}, \mathcal{X}_{\text{Dobs}}, N_s, N_e$)

1 $\mathcal{T} \leftarrow \boldsymbol{x}_s$; $\boldsymbol{x}_{\text{root}} \leftarrow \boldsymbol{x}_s$;
2 $\mathcal{S} \leftarrow$ `SampleFree`$(N) \cup \boldsymbol{x}_s \cup \boldsymbol{x}_g$;
3 $\mathcal{V}_b \leftarrow \emptyset$; $\mathcal{Q}_o \leftarrow \emptyset$; $\mathcal{Q}_r \leftarrow \emptyset$;
4 $\mathcal{V}_{unv} \leftarrow \mathcal{S} \backslash \{\boldsymbol{x}_s\}$; $\mathcal{V}_{\text{open}} \leftarrow \{\boldsymbol{x}_s\}$; $\mathcal{V}_{\text{closed}} \leftarrow \emptyset$;
5 $\boldsymbol{z} \leftarrow \boldsymbol{x}_s$;
6 **while** *True* **do**
7 $(\boldsymbol{x}_{\text{robot}}, \boldsymbol{x}_g, \mathcal{N}_b) \leftarrow$ `UpdateContext`$(\mathcal{T}, \mathcal{X}_{\text{Dobs}})$;
8 **for** $i = 1$ **to** N_e **do**
9 `ExpandFMT`$(\mathcal{T})$;
10 `RewireFromObstacles`$(\mathcal{T})$;
11 `RewireFromRoot`$(\mathcal{T})$;
12 $(\boldsymbol{x}_{\text{root}}, \boldsymbol{x}_1, ..., \boldsymbol{x}_k) \leftarrow$ `GeneratePath`$(\mathcal{T}, \boldsymbol{x}_{\text{root}})$;
13 **if** $\boldsymbol{x}_{\text{robot}}$ is near $\boldsymbol{x}_{\text{root}}$ **then**
14 $\boldsymbol{x}_{\text{root}} \leftarrow \boldsymbol{x}_1$;
15 `UpdateRoot`$(\mathcal{T}, \boldsymbol{x}_{\text{root}})$;
16 Steer robot towards $\boldsymbol{x}_{\text{root}}$;
17 Perform other tasks;

velocity commands at specific intervals. Therefore, we only allow the tree expansion and rewiring to run for a defined number of iterations (N_e) to expand and rewire the tree. This method can also be easily adapted to run for a desired time interval or planning frequency.

The method is shown in Algorithm 1, which starts by sampling N configurations in free space (Line 2) considering only the fixed obstacles. This function also calculates the neighborhood radius r_n based on the number of samples and the dimensionality of the problem according to

$$r_n = \gamma_s 2\left(1 + \frac{1}{d}\right)^{\frac{1}{d}} \left(\frac{\mu(\mathcal{X}_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}} \left(\frac{\log(N)}{N}\right)^{\frac{1}{d}}, \quad (1)$$

where $\gamma_s > 1$ is a tuning parameter, d is the dimension of the problem, $\mu(\mathcal{X}_{\text{free}})$ is the Lebesgue measure of the free space, and ζ_d is the volume of a unit ball in $\mathbb{R}^d$. Although FMT* provides an equation for r_n, we use the equation defined in [7] for PRM* since it computes an r_n slightly bigger than the FMT* equation for r_n.

Inside the infinite loop, the algorithm updates the context of the problem by returning the current position of the robot and goal x_g, and by finding the nodes in the tree that have been blocked or unblocked by the dynamic obstacles. When a configuration $\boldsymbol{x}_q$ in the tree is blocked, its cost $\text{c}(\boldsymbol{x}_q)$ is set to infinity. When a node is unblocked, its cost is updated to

$$\text{c}(\boldsymbol{x}_q) = \text{c}(\boldsymbol{x}_{\text{parent}}) + \text{Cost}(\boldsymbol{x}_{\text{parent}}, \boldsymbol{x}_q), \quad (2)$$

where $\boldsymbol{x}_{\text{parent}}$ is the configuration of the parent of $\boldsymbol{x}_q$, and

$$\text{Cost}(\boldsymbol{u}, \boldsymbol{v}) = ||\boldsymbol{v} - \boldsymbol{u}||. \quad (3)$$

If the updated node has children, its children's costs are recursively updated. Lines 9–11 expand the tree according to Algorithm 2, and rewire the tree based on Algorithm 3. Line 12 returns a global path if $\boldsymbol{x}_g$ is in the tree or a local path otherwise. The local path is found by computing

$$\boldsymbol{x}_k \leftarrow \arg\min_{\boldsymbol{x} \in \mathcal{T}} \text{c}(\boldsymbol{x}) + ||\boldsymbol{x} - \boldsymbol{x}_{goal}||, \quad (4)$$

and then a path starting at x_{root} and ending at $\boldsymbol{x}_k$ is generated. While we do not limit k, RT-RRT* generates a path up to a specific k. More details on the local path generation can be found in Algorithm 6 in Naderi et al. [5]. The approach resembles the A* search.

In Line 15, the root of the tree is set to the next configuration in the path, which is always the second element since the path always starts at the old root. Finally, the algorithm steers the robot towards the new root of the tree on Line 16. If there are other tasks to perform such as mapping, and localization, they can be called in the main function as well.

A. Expanding the Tree

The tree expansion is described in Algorithm 2. Most of the algorithm (Lines 1–14) was inspired by FMT*, proposed in [8], but our implementation has two major differences.

First, the loops in the original implementation were substituted for conditional statements. These statements ensure that only one node can be added in the tree per call. If multiple nodes are added to the tree at once, there is a possibility of delaying other tasks since the processor will spend too much time expanding many nodes at once. In addition, Line 6 not only checks for fixed obstacles but also checks whether $\boldsymbol{y}_{\min}$ is not being blocked by a dynamic obstacle according to (2).

Second, in the original approach, once a node has checked all possible connections with its neighbors, it is closed and never checked again. In our approach, we do not spend time expanding the nodes that are inside $\mathcal{X}_{\text{Dobs}}$. As a consequence, when a dynamic obstacle moves, there will be unvisited nodes around closed nodes. To add these nodes to the tree, our algorithm must be able to reopen closed nodes nearby. This is done by adding all $\boldsymbol{z}$ that are near unvisited nodes at closing time to $\mathcal{V}_{\text{toOpen}}$ (Line 15). Then, when the regular expansion is finished (Line 17), the algorithm reopens these nodes (Line 18) to continue the expansion in case a dynamic obstacle moves.

B. Rewiring the Tree

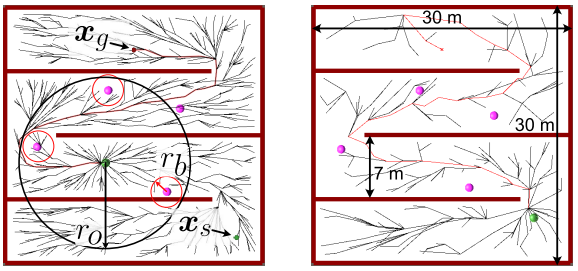
As the dynamic obstacles move around the environment, the tree nodes are constantly being blocked and unblocked by Line 7 in Algorithm 1. When a node is blocked or unblocked, its cost is changed and all its children are recursively updated. The task of the `RewireFromObstacle` method in Algorithm 3 is to find the connections with lower cost in the neighborhood of nodes that have recently been blocked or unblocked. This function only rewires the nodes that have recently been affected by a dynamic obstacle. The rewiring process starts by adding all blocked nodes to $\mathcal{Q}_o$. Then, nodes are iteratively removed from the list and the algorithm tries to find parents nearby with a lower cost. If there is a connection with a lower cost that is also collision-free, the children of the updated nodes are also added to $\mathcal{Q}_o$.

Algorithm 2: ExpandFMT($\mathcal{T}$)

1 **if** $\mathcal{X}_{near} = \emptyset$ & $\boldsymbol{z} \neq \emptyset$ **then** $\mathcal{X}_{near} \leftarrow$ `Near` $(\boldsymbol{z}, \mathcal{V}_{unv})$;
2 **else**
3 $\quad \boldsymbol{x} =$ `PopLast`$(\mathcal{X}_{near})$;
4 $\quad \mathcal{Y}_{near} \leftarrow$ `Near` $(\boldsymbol{x}, \mathcal{V}_{\text{open}})$;
5 $\quad y_{\min} \leftarrow \arg\min_{\boldsymbol{y} \in \mathcal{Y}_{near}} (\texttt{c}(\boldsymbol{y}) + \texttt{Cost}(\boldsymbol{x}, \boldsymbol{y}))$;
6 $\quad$ **if** `CollisionFree`$(\boldsymbol{y}_{\min}, \boldsymbol{x}) \& \texttt{c}(\boldsymbol{y}_{\min}) < \infty$ **then**
7 $\quad\quad \mathcal{V}_{open,new} \leftarrow \mathcal{V}_{open,new} \cup \{\boldsymbol{x}\}$;
8 $\quad\quad \mathcal{V}_{unv} \leftarrow \mathcal{V}_{unv} \backslash \{\boldsymbol{x}\}$;
9 $\quad\quad \texttt{c}(\boldsymbol{x}) \leftarrow \texttt{c}(\boldsymbol{y}_{\min}) + \texttt{Cost}(\boldsymbol{y}_{\min}, \boldsymbol{x})$;
10 $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\boldsymbol{x}, (\boldsymbol{y}_{\min}, \boldsymbol{x})\}$
11 $\quad$ **if** $\mathcal{X}_{near} = \emptyset$ & $\boldsymbol{z} \neq \emptyset$ **then**
12 $\quad\quad$ `Close` $(\boldsymbol{z})$;
13 $\quad\quad \mathcal{Z}_{near} \leftarrow$ `Near` $(\boldsymbol{z}, \mathcal{V}_{unv})$;
14 $\quad\quad \boldsymbol{z} \leftarrow \arg\min_{\boldsymbol{y} \in \mathcal{V}_{\text{open}}} \texttt{c}(\boldsymbol{y})$;
15 $\quad\quad$ **foreach** $\boldsymbol{x} \in \mathcal{Z}_{near}$ **do**
16 $\quad\quad\quad$ **if** `CollisionFree` $(\boldsymbol{z}, \boldsymbol{x})$ **then**
$\quad\quad\quad\quad \mathcal{V}_{\text{toOpen}} \leftarrow \mathcal{V}_{\text{toOpen}} \cup \boldsymbol{z}$;
17 $\quad$ **if** $\boldsymbol{z} = \emptyset$ **then**
18 $\quad\quad$ `Open`$(\mathcal{V}_{\text{toOpen}})$; $\mathcal{V}_{\text{toOpen}} = \emptyset$;
19 $\quad\quad \boldsymbol{z} \leftarrow \arg\min_{\boldsymbol{y} \in \mathcal{V}_{\text{open}}} \texttt{c}(\boldsymbol{y})$;
20 **def** `Close` $(\mathcal{V})$**:**
21 $\quad \mathcal{V}_{\text{open}} \leftarrow \mathcal{V}_{\text{open}} \cup \mathcal{V}_{open,new} \backslash \mathcal{V}$;
22 $\quad \mathcal{V}_{\text{closed}} \leftarrow \mathcal{V}_{\text{closed}} \cup \mathcal{V}$;
23 **def** `Open` $(\mathcal{V})$**:**
24 $\quad \mathcal{V}_{\text{open}} \leftarrow \mathcal{V}_{\text{open}} \cup \mathcal{V}$;
25 $\quad \mathcal{V}_{\text{closed}} \leftarrow \mathcal{V}_{\text{closed}} \backslash \mathcal{V}$;

Algorithm 3: RewireFromObstacles($\mathcal{T}$)

1 **if** $\mathcal{Q}_o = \emptyset$ **then** $\mathcal{Q}_o \leftarrow \mathcal{N}_b$;
2 **else**
3 $\quad \boldsymbol{x}_b \leftarrow$ `PopFirst`$(\mathcal{Q}_o)$;
4 $\quad$ **if** $\boldsymbol{x}_b \notin \mathcal{X}_{\text{Dobs}}$ **then**
5 $\quad\quad \mathcal{Y}_{near} \leftarrow$ `Near`$(\boldsymbol{x}_b, \mathcal{V}_{\text{open}} \cup \mathcal{V}_{\text{closed}})$;
6 $\quad\quad y_{\min} \leftarrow \arg\min_{\boldsymbol{y} \in \mathcal{Y}_{near}} (\texttt{c}(\boldsymbol{y}) + \texttt{Cost}(\boldsymbol{x}_b, \boldsymbol{y}))$;
7 $\quad\quad$ **if** `CollisionFree`$(\boldsymbol{y}_{\min}, \boldsymbol{x}) \& \texttt{c}(\boldsymbol{y}_{\min}) < \infty$ **then**
8 $\quad\quad\quad$ `UpdateParentChild`$(\mathcal{T}, \boldsymbol{y}_{\min}, \boldsymbol{x}_b)$;
9 $\quad\quad\quad$ `RecalculateChildrenCost`$(\boldsymbol{x}_b)$



(a) RT-FMT (b) RT-RRT*

Fig. 2: Simulation environment with the Maze space for (a) RT-FMT and (b) RT-RRT*. The parameter r_o represents the sensing range of the robot, and the distance r_b represents the safety radius in which tree nodes are considered blocked if a dynamic obstacle is within r_o from the robot.

As the robot moves around the environment, the tree root is updated by Algorithm 1, Line 15. When this happens, the algorithm triggers the `RewireFromRoot` function (Line 11) to rewire all nodes in the tree by inserting the new root into $\mathcal{Q}_r$. The `RewireFromRoot` function is very similar to Algorithm 3. The algorithm removes a node from $\mathcal{Q}_r$ and tries to find better connections in the tree. When a new connection is made, the children of the updated node are also added to $\mathcal{Q}_r$. This causes a chain reaction that starts from the root and updates all nodes in the tree. This method has also been implemented without any loops to only update a single node per call. Our implementation triggers this chain reaction whenever the root is updated. However, this can easily be modified to happen at a desired frequency.

IV. EVALUATION METHODOLOGY

The main goal of our evaluation process was to study the advantages of real-time planning and execution using FMT over RRT in certain environments, as well as to verify whether real-time execution and planning can result in faster arrival times. To properly achieve our evaluation goals, we implemented and tested our algorithm in Unity 3D using a similar environment to the one provided by [5]. Then, we prepared two different scenarios. A Maze space, shown in Fig. 2, which was designed as a toy problem to highlight one shortcoming of RRT-based algorithms in maze-like spaces. This shortcoming is caused by the lower probability of successfully connecting a node to the tree without hitting a wall. The second environment is a Mine-like space illustrated in Fig. 1, as an application problem that was designed following the room and pillar method of creating underground mines.

The experiments in the Maze space were executed with the robot traveling at 2 m/s, with $r_b = 2$ m, $r_o = 10$ m, and a robot radius of 0.5 m. For the Mine space, the robot traveled at 4 m/s, with $r_b = 14$ m, $r_o = 50$ m, and a robot radius of 1.5 m. For both environments, the dynamic obstacles traveled at half the robot's speed. All remaining parameters of RT-RRT* were kept the same as provided by [5]. F or example, the number of tentative expansions per action (N_e) was set to 32. For RT-FMT, we also maintained $N_e = 32$, and γ_s, defined in (1), was kept at 1.1 for all experiments.

We split the evaluation methodology into three experiments with each one repeated 50 times for a different number of sample attempts. They ranged from 500 to 4500 samples with an increment of 1000 samples. It is important to emphasize that sample attempts is not the number of nodes in the tree. For RT-FMT, it indicates the free space sample count, and for RT-RRT*, the sample attempts represent the number of times the algorithm sampled a point and attempted

to expand the tree. Since both approaches are very different in building the tree, care must be taken when analyzing the results. This same approach was applied by [8] to compare FMT* against RRT*.

Experiment 1 involved obtaining the planning times, executed costs, and arrival times by running the planners in a non real-time manner; i.e., allowing the robot to move only after trying the defined number of samples and finding the complete path. These metrics were used as a baseline to judge the performance of the real-time execution for Experiments 2 and 3. Experiment 2 allowed the robots to execute the local paths in real-time without the presence of dynamic obstacles. With this experiment, it is possible to evaluate whether the robots can arrive earlier at the goal, considering that there is a risk of finding a costlier trajectory when compared to Experiment 1. Experiment 3 is similar to 2, but with dynamic obstacles present in the space. In the Maze space, the dynamic obstacles maintained a fixed initial position and they moved in random directions. In the Mine space, their vertical positions were randomly selected at the start of the simulation, and they would move down if they appeared in the top part of the environment, and vice-versa.

The RT-FMT implementation used for the evaluation procedure is available at github.com/offroad-robotics/rt-fmt-icra.git.

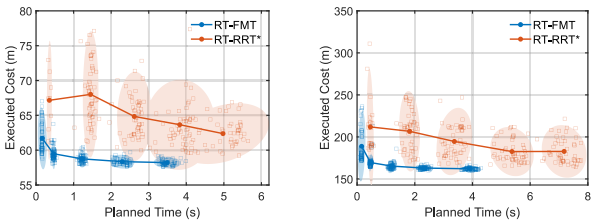
V. SIMULATED EXPERIMENTS

This section presents the results obtained from the simulations of the different experiments and discusses the results. During the experiments, both algorithms reached quickly 100 % success rate on both spaces during experiments 1 and 2 when increasing the number of sample attempts. During Experiment 3, which is the more challenging one, only RT-FMT reached 100 % at 4500 samples in the Maze space, while RT-RRT* achieved only 46 % at the same number of samples. For the Mine space, RT-FMT achieved 84 % and RT-RRT only 50 % success rate. The low success rate of RT-RRT* is caused by node connections with longer edges than RT-FMT, as shown in Fig. 2.

During the execution of both algorithms, the robot does not react to changes in the environment when it is moving in between nodes. While it is possible to limit the edge lengths of RT-RRT* to improve the success rate of the algorithm, it worsens other important characteristics such as planning time and executed cost that are analyzed in this section. For such reasons, we did not modify this behavior in RT-RRT*.

A. Experiment 1

For this experiment, we show the solution cost for all runs in Fig. 3. It is clear that RT-FMT outperforms RT-RRT* in planning time and solution cost for both environments. The solution was found faster with the blue curve located to the left side of the graph (shorter planning time) and below the orange line (shorter executed path). Also, the solution found by the RT-FMT yielded a shorter path for the same sample count. This is because, for the same number of sample count (i.e., attempts to connect points to the tree), RT-FMT will



(a) Maze space (b) Mine space

Fig. 3: Simulation results for Experiment 1. Hollow squares represent a single simulation, solid circles the average on both axes for different sample counts, and the ellipses 90 % confidence level.

have many more nodes added to the tree than RT-RRT*, which is a direct consequence of how the samples of RRT* and FMT* are computed.

Furthermore, the planning time is much more predictable in RT-FMT than RT-RRT* due to its smaller deviation, which indicates that RT-FMT is better suited for real-time applications due to the hard real-time bounds that are present in these systems [6].

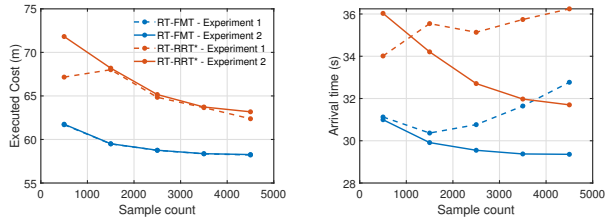
B. Experiment 2

For the results of this experiment and Experiment 3, instead of displaying the planning time as in Fig. 3, we display the sample count in the x-axis because it is fixed for each simulation group. The results of this experiment are shown in Fig. 4 and Fig. 5 for the Maze and Mine spaces, respectively. From these images, we can draw a few relevant conclusions. First, the executed cost and arrival time of RT-FMT was also smaller than RT-RRT* for both environments. And, interestingly, RT-FMT was able to match the ideal cost (from Experiment 1) almost perfectly, as can be seen in Fig. 4a and Fig. 5a. RT-RRT* also performed relatively well.

Another relevant observation to mention in Fig. 4b and Fig. 5b is the fact that the arrival time for Experiment 1 starts to monotonically increase after 1500 samples. This is caused by the extra time the algorithm takes to expand the tree with more samples. As a consequence of this fact and of both algorithms being able to closely match the executed costs to Experiment 1, both approaches had smaller arrival times for higher sample counts (dashed lines in Fig. 4b and Fig. 5b). This result indicates that it is indeed worthwhile risking taking a sub-optimal path, in the beginning, to save planning time since the difference in executed cost was small for RT-RRT* and negligible for RT-FMT.

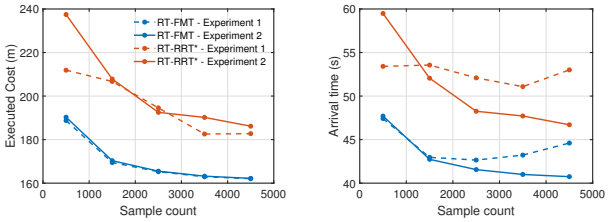
C. Experiment 3

After adding the dynamic obstacles, the cost of the executed path, unsurprisingly, increased with respect to the execution without obstacles (Experiment 2). This is shown in both Fig. 6 and Fig. 7. However, that was the only common characteristic between the results of the Maze and Mine spaces for Experiment 3.



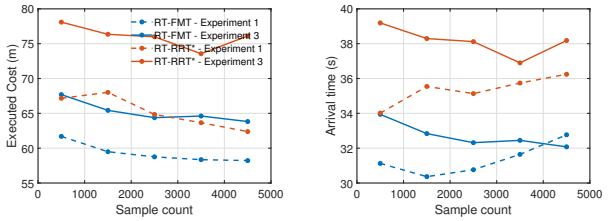
(a) (b)

Fig. 4: Simulation results for the Maze in Experiment 2.



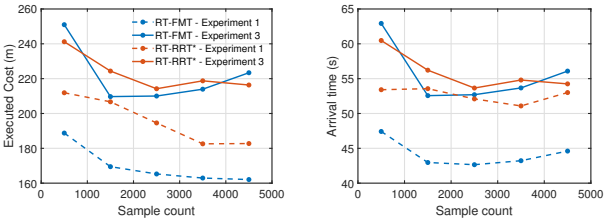
(a) (b)

Fig. 5: Simulation results for the Mine in Experiment 2.



(a) (b)

Fig. 6: Simulation results for the Maze in Experiment 3.



(a) (b)

Fig. 7: Simulation results for the Mine in Experiment 3.

If we focus on the arrival time of the maze space for RT-FMT (Fig. 6b), we see that as the number of samples increased, the difference in arrival time of Experiment 3 and Experiment 1 became smaller up to the point that the real-time execution arrived faster at the goal state, even with the extra task of avoiding dynamic obstacles. This is an excellent result since it is very common to select a very high number of nodes to guarantee that a solution will be found.

Regarding the Mine space, the big difference in executed cost of Experiment 3 and Experiment 1 in Fig. 7 can be explained by looking at the layout of the mine in Fig. 1. When a dynamic obstacle blocks the robot in the middle of a hallway, the robot has to go around the block to avoid it, which greatly increases the executed cost. That is also the reason why RT-FMT and RT-RRT* performed similarly, they both had to go around the same number of blocks, on average, to avoid the obstacles.

Another behavior that is important to discuss is the upward trend of the arrival cost as the number of samples increased in Fig. 7b for RT-FMT. This was a consequence of the Mine space layout, the sample count definition in Section IV, and how RT-FMT and RT-RRT* sample points in the environment. The Mine space contains more occupied space than free space, which caused the RT-FMT samples to be densely packed since they are guaranteed to lie in the small free space. In contrast, most sampled points for RT-RRT* failed to connect to the tree because of the likelihood of collisions. As a result, RT-FMT had many more nodes in the tree than RT-RRT*. With many nodes densely packed, the rewiring process of RT-FMT became too slow to respond to the dynamic obstacles in advance, causing the robot to react to dynamic obstacles as if it was short-sighted, which, consequently, also forced the robot to take longer routes. The same effect would have happened to RT-RRT* if the tree had a similar number of nodes to RT-FMT. Still, RT-FMT had an 84 % success rate while RT-RRT* had only 50 %.

VI. CONCLUSIONS

This paper proposed RT-FMT, which is a real-time path planning algorithm that is capable of quickly generating low-cost paths in environments with dynamic obstacle avoidance with unknown trajectories. The real-time capability of the algorithm comes from the fact that it generates local paths for the robot to follow while the global path is not available. All these characteristics indicate that RT-FMT is ideal for time-critical applications. To show the capabilities of RT-FMT, we compare it against RT-RRT* on simulated environments with dynamic obstacles. The results show that RT-FMT has higher success rates, lower traveled distances, and smaller arrival times in almost all cases. The greater performance of RT-FMT over RT-RRT* is mostly related to the fact that, in cluttered spaces, the probability of randomly connecting a node to the tree is very low for RT-RRT*, which requires many attempts to grow the tree. On the other hand, RT-FMT starts the search with all nodes already laid out in the environment, which reduces the number of attempts to expand the tree.

Future work on the proposed algorithm will involve an implementation in the Open Motion Planning Library (OMPL) [13] to facilitate comparisons with other state-of-the-art approaches, mainly on higher dimensional problems, and to understand the effects of the real-time variation on the complexity of the algorithm and its computing time. The OMPL implementation might also simplify tests on real robots. In addition, a variation of the approach that works for constrained robots can also be implemented.

REFERENCES

[1] F. Tian, R. Zhou, Z. Li, L. Li, Y. Gao, D. Cao, and L. Chen, "Trajectory planning for autonomous mining trucks considering terrain constraints," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 4, pp. 772–786, 2021.

[2] S. Hayat, E. Yanmaz, C. Bettstetter, and T. X. Brown, "Multi-objective drone path planning for search and rescue with quality-of-service requirements," *Autonomous Robots*, vol. 44, no. 7, pp. 1183–1198, 2020.

[3] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 1478–1483.

[4] J. Xu, K. Song, D. Zhang, H. Dong, Y. Yan, and Q. Meng, "Informed anytime fast marching tree for asymptotically optimal motion planning," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 6, pp. 5068–5077, 2020.

[5] K. Naderi, J. Rajamäki, and P. Hämäläinen, "RT-RRT*: a real-time path planning algorithm based on RRT," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, 2015, pp. 113–118.

[6] B. Cserna, M. Bogochow, S. Chambers, M. Tremblay, S. Katt, and W. Ruml, "Anytime versus real-time heuristic search for online planning," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 7, no. 1, 2016.

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[8] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

[9] F. Grothe, V. N. Hartmann, A. Orthey, and M. Toussaint, "St-rrt*: Asymptotically-optimal bidirectional motion planning through space-time," in *Proceedings of the 2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 3314–3320.

[10] B. D. Luders, S. Karaman, E. Frazzoli, and J. P. How, "Bounds on tracking error using closed-loop rapidly-exploring random trees," in *Proceedings of the 2010 American Control Conference*. IEEE, 2010, pp. 5406–5412.

[11] B. Chandler and M. A. Goodrich, "Online rrt and online fmt: Rapid replanning with dynamic cost," in *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 6313–6318.

[12] B. Tong, Q. Liu, and C. Dai, "A rrt* fn based path replanning algorithm," in *Proceedings of the 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 1. IEEE, 2019, pp. 1435–1445.

[13] I. A. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012.