Implementation of the CAF-RRT* for a mobile robot

1st Jonathan Crespo
Robotics Engineering
University of Maryland
College Park,
jcrespo@umd.edu

2nd Naga Kambhampati
Robotics Engineering
University of Maryland
College Park, USA
saigopal@umd.edu

3rd Gaurav Upadhyay
Robotics Engineering
University of Maryland
College Park, USA
Ugaurav@umd.edu

***Abstract*—The authors present CAF-RRT*, a novel algorithm designed to enhance path planning for robots in two-dimensional spaces. This algorithm builds upon the existing Rapidly-exploring Random Trees (RRT) framework by integrating the Quick-RRT* with a bidirectional RRT method, allowing for rapid generation of an initial path that serves as a basis for further refinements. The optimization of the path leverages a unique strategy that employs triangle geometry to effectively reduce costs. Additionally, a circular arc fillet method is implemented to smooth the path, thereby enhancing its safety and smoothness. The Script with python and created a simulation maze for a 2D representation of a turtlebot 3 and another simulation in a more complex maze considering a point robot.**

Index Terms*—RRT, RRT

I. Introduction

Path planning is a foundational issue in robotics, relevant to applications ranging from robot navigation and autonomous vehicles to unmanned aerial vehicles. It involves navigating a robot from a start point to a destination through an obstacle-laden environment. Typically, this problem is tackled using either grid-based search algorithms like A*, or sampling-based methods such as Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM), which rely on extensive random sampling. In theory, these methods can achieve a solution probability of 100 percent with infinite iterations.

RRT, The Rapidly-exploring Random Tree (RRT) algorithm is a widely used path planning method in robotics that helps a robot navigate from a start point to a target by growing a tree structure through random sampling of the environment. This technique enables efficient exploration of complex spaces, making it suitable for navigating around obstacles. However, its paths often contain sharp turns and higher path costs, requiring further optimization or smoothing.

Bidirectional Rapidly-exploring Random Tree (RRT) improves path planning by simultaneously growing two trees: one from the start and the other from the goal. The trees advance toward each other until they meet, creating a complete path. This approach speeds up pathfinding by reducing exploration time. However, the resulting paths may still need further optimization or smoothing, as they often have sharp turns or high costs.

RRT*, is an optimized version of the RRT pathfinding algorithm that focuses on finding more efficient paths. While it starts by expanding a tree-like structure through random sampling similar to RRT, RRT* improves the process by continuously optimizing the tree structure as more samples are added. It selects parent nodes that minimize path costs and rewires the tree to ensure that paths are progressively shortened. This process reduces sharp turns and improves overall path smoothness and efficiency. Despite slower convergence than RRT due to its additional calculations, RRT* is well-suited for finding near-optimal paths in environments with obstacles or complex terrain.

Quick-RRT*, is a modified version of the RRT* algorithm that aims to improve convergence speed by expanding the set of candidate parent nodes. It uses depth parameters and principles of triangle inequality to consider not just immediate neighbors, but also ancestor nodes. By identifying a broader pool of parent nodes when adding new branches, Quick-RRT* reduces path costs and accelerates convergence.

RRT*-AB, enhances the efficiency of RRT* by locating the target region quickly through target-biased bounded sampling. After identifying the initial path, it optimizes the route by removing unnecessary nodes and refining it through centralized bounded sampling.

CAF-RRT*, enhances RRT* by combining the Quick-RRT* method with a bidirectional approach to efficiently create an initial path. It optimizes this path using triangle geometry, while a circular arc fillet technique smooths sharp corners and reduces path costs. This approach improves safety, efficiency, and navigational performance compared to standard RRT algorithms.

II. Proposed Method

This section first outlines the variables, symbols, and functions utilized by the algorithm, followed by a detailed presentation of the overall framework of the proposed method. It includes the pseudo-code for implementing the algorithm. The latter part of the section, divided into subsections C and D, delves into the two primary innovations introduced in this paper: the strategy for path optimization and the technique for path smoothing, which utilizes the triangle rule.

A. Definition of the Problem

The configuration space is denoted as $X = (0,1)^d$, with X_{obs} indicating the areas filled with obstacles. The newly extended node is referred to as x_{new}. The region near obstacles likely to cause collisions is labeled Xp_{obs}. The free space, X_{free}, is the complement of X_{obs}, and the set $(X_{\text{free}}, x_{\text{init}}, X_{\text{goal}})$ establishes the path planning challenge, where $x_{\text{init}} \in X_{\text{free}}$ is the starting point, and $X_{\text{goal}} \subseteq X_{\text{free}}$ is the destination.

The collections of paths generated at different stages of the algorithm are stored in T_{init} for initial paths, T_{opt} for optimized paths, and T_{final} for the final smoothed paths. A path is collision-free if it holds that for every $\tau \in [0,1]$, $\sigma(\tau) \in X_{\text{free}}$. The optimal path, σ^*, is the one that minimizes $c(\sigma)$, subject to $\sigma(0) = x_{\text{init}}$, $\sigma(1) \in X_{\text{goal}}$, and $\sigma(\tau) \in X_{\text{free}}$ for all $\tau \in [0,1]$.

Definition 1: The safety distance is such that for any point x on the path $\sigma(\tau)$, for any $x_{\text{obs}} \in X_{\text{obs}}$, the condition $\min \|x - x_{\text{obs}}\|^2 > q$ is satisfied.

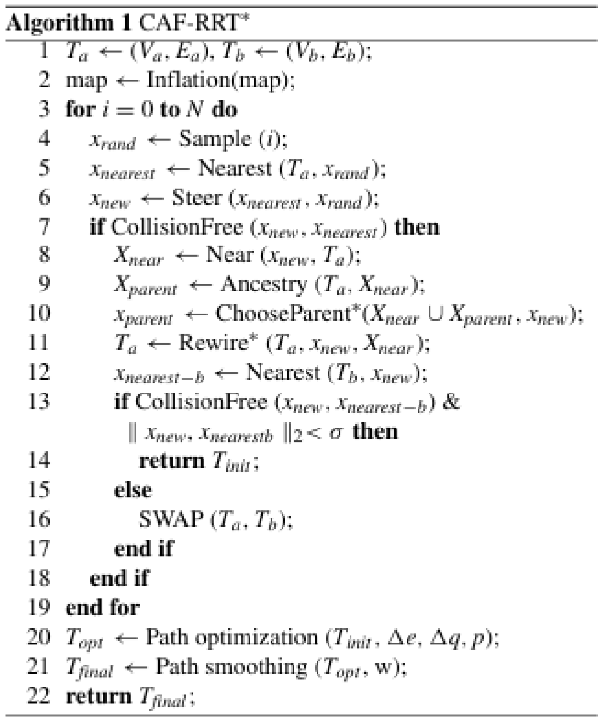
Definition 2: The collision probability area for x_{new} is defined such that if $\|x_{\text{new}} - X_{\text{obs}}\|^2 < q$, then x_{new} is deemed to have entered Xp_{obs}.

The document also outlines functions used in the pseudocode. N represents the number of nodes in T_{init} or T_{opt}. V is a set of points in these trees. The function Angle calculates the angle between two points relative to the coordinate axis using the arctan function, and the function Insert is used to add a node between two points in the structure.

B. The Main Framework

The algorithm begins by enlarging obstacles with an inflation strategy for safer path planning. Using bidirectional RRT and heuristic search, it rapidly establishes an initial path, though with many intermediate nodes. To mitigate this, the approach adopts Quick-RRT*'s method of selecting parent nodes through depth parameters, which streamlines path creation and reduces collision checks, thereby enhancing runtime efficiency.

Subsequent to the initial path setup, it undergoes optimization using the triangle rule to adjust paths for cost-efficiency, detailed in subsection C.The path is further smoothed with a circular arc fillet method to ensure a smoother trajectory and lower energy consumption, as outlined in subsection D. The core processes, including functions like Ancestry, ChooseParent, and Rewire, are encapsulated in the pseudocode of Algorithm 1 and work as Quick-RRT* states, with comprehensive details provided in the referenced literature.



Algorithm 1 CAF-RRT*

```
$T_a \leftarrow (V_a, E_a), T_b \leftarrow (V_b, E_b)$;
map $\leftarrow$ Inflation(map);
for $i = 0$ to $N$ do
    $x_{rand} \leftarrow$ Sample $(i)$;
    $x_{nearest} \leftarrow$ Nearest $(T_a, x_{rand})$;
    $x_{new} \leftarrow$ Steer $(x_{nearest}, x_{rand})$;
    if CollisionFree $(x_{new}, x_{nearest})$ then
        $X_{near} \leftarrow$ Near $(x_{new}, T_a)$;
        $X_{parent} \leftarrow$ Ancestry $(T_a, X_{near})$;
        $x_{parent} \leftarrow$ ChooseParent*$(X_{near} \cup X_{parent}, x_{new})$;
        $T_a \leftarrow$ Rewire* $(T_a, x_{new}, X_{near})$;
        $x_{nearest-b} \leftarrow$ Nearest $(T_b, x_{new})$;
        if CollisionFree $(x_{new}, x_{nearest-b})$ &
          $\| x_{new}, x_{nearestb} \|_2 < \sigma$ then
            return $T_{init}$;
        else
            SWAP $(T_a, T_b)$;
        end if
    end if
end for
$T_{opt} \leftarrow$ Path optimization $(T_{init}, \Delta e, \Delta q, p)$;
$T_{final} \leftarrow$ Path smoothing $(T_{opt}$, w$)$;
return $T_{final}$;
```

Fig. 1. Pseudo code of the algorithm

C. Path Optimization

Once the initial path is created by integrating Quick-RRT with bidirectional RRT, a path optimization approach involving alternating equal distance and equal proportion strategies is implemented. This method effectively reduces the path cost while maintaining safety. The procedure involves sequentially selecting each intermediate node and its two neighboring edges to form triangles for optimization. For instance, with any intermediate node V_i, the nodes immediately preceding and following it (V_{i-1} and V_{i+1}) are used to form triangles referred to as $V_{i-1}V_iV_{i+1}$.

In the equal-distance strategy, a predefined length e is used. Starting from V_i to V_{i-1}, a point V_{i-1} is placed at every e interval, and similarly from V_i to V_{i+1}. This forms a new triangle $V_{i-1}V_iV_{i+1}$. In the equal-proportion approach, a ratio p (where p is between 0 and 1) is defined. Points are placed at a distance of $V_iV_{i-1} \times p$ and $V_iV_{i+1} \times p$, to construct another triangle named $V_{i-1}V_iV_{i+1}$.

Collision detection is performed on edges $V_{i-1}V_{i+1}$ or $V_{i-1}V_{i+1}$ to ensure that the selected nodes comply with the security policies. The triangle rule is applied multiple times to further decrease path costs. Due to frequent segmentations on the line segment, excess nodes might be generated, which should be removed to avoid compromising the efficiency of the subsequent smoothing process using the fillet method. The detailed steps of this method are illustrated in the pseudocode of Algorithm 2, and the efficacy of these techniques is demonstrated in Figure below.

Algorithm 2 Path Optimization ( )

Input: T_{init}, Δe, Δq, p

Output: T_{opt}

```
for $j = 1$ to 2 do
  for $i = 2$ to $N_{init} - 1$ do
    $l_1 = \| V_i, V_{i-1} \|_2$; $l_2 = \| V_i, V_{i+1} \|_2$;
    $\alpha_1 = \text{Angle}(V_i, V_{i-1})$; $\alpha_2 = \text{Angle}(V_i, V_{i+1})$;
    $V'_{(i-1)x} = V_{ix} + \Delta e \times \cos(\alpha_1)$; $V'_{(i-1)y} = V_{iy} + \Delta e \times \sin(\alpha_1)$;
    $V'_{(i+1)x} = V_{ix} + \Delta e \times \cos(\alpha_2)$; $V'_{(i+1)y} = V_{iy} + \Delta e \times \sin(\alpha_2)$;
    if CollisionFree ($V'_{i-1}, V'_{i+1}$) then
      delete $V_i$;
      $(V_{i-1}, V_{i+1}) \leftarrow \text{Insert}(V'_{i-1}, V'_{i+1})$;
    end if
  end for
  return $T_{opt}$;
  for $i = 2$ to $N_{opt} - 1$ do
    $l_1 = \| V_i, V_{i-1} \|_2$; $l_2 = \| V_i, V_{i+1} \|_2$;
    $\alpha_1 = \text{Angle}(V_i, V_{i-1})$; $\alpha_2 = \text{Angle}(V_i, V_{i+1})$;
    $V''_{(i-1)x} = V_{ix} + l_1 \times p \times \cos(\alpha_1)$; $V''_{(i-1)y} = V_{iy} + l_1 \times p \times \sin(\alpha_1)$;
    $V''_{(i+1)x} = V_{ix} + l_2 \times p \times \cos(\alpha_2)$; $V''_{(i+1)y} = V_{iy} + l_2 \times p \times \sin(\alpha_2)$;
    if CollisionFree ($V''_{i-1}, V''_{i+1}$) then
      delete $V_i$;
      $(V_{i-1}, V_{i+1}) \leftarrow \text{Insert}(V''_{i-1}, V''_{i+1})$;
    end if
  end for
  return $T_{opt}$;
end for
for $k = 2$ to $N_{opt} - 1$ do
  if CollisionFree ($V_{k-1}, V_{k+1}$) then
    delete $V_k$;
    Update $T_{opt}$;
  end if
end for
return $T_{opt}$;
```

Fig. 2. Pseudo code of the algorithm

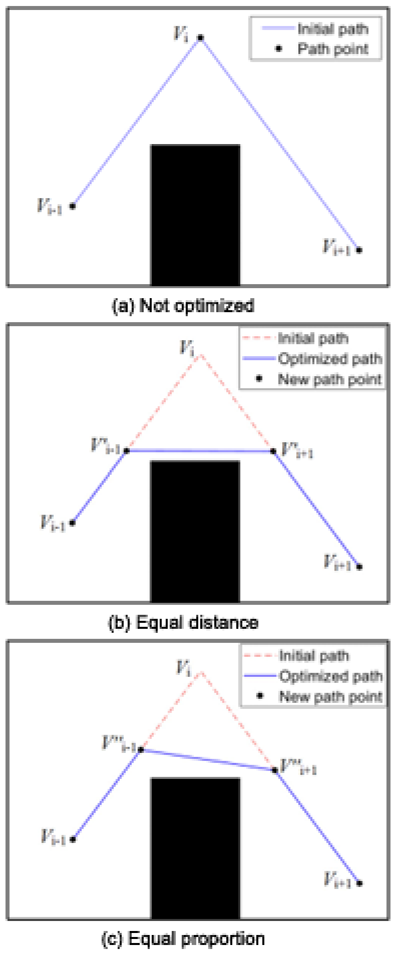


Fig. 3. Optimized Path

D. Path Smoothing

Following the path optimization phase, the cost of the resulting path is greatly minimized. Nonetheless, the path remains rough around the edges, which calls for further refinement to elevate the overall quality of the path. In this study, we employ a path smoothing technique based on the circular arc fillet method, suited for comprehensive path planning efforts. Notably utilized in both engineering and architectural contexts, the circular arc fillet method stands out for its versatile geometric capabilities, allowing it to handle straight lines intersecting at any angle. This method effectively smooths paths by converting linear segments into curved arcs, as detailed in the subsequent sections of this paper.

After smoothing the given path, all sharp edges should be eliminated, which is contingent not only on the circle's radius but also on where the cuts are made. Figure 5 (a) displays two circles along the same segment of the sub-path, each with the same radius but different impacts on the path's smoothness due to their respective cutting points. After making cuts based on circles C and D, new sub-paths emerge, as depicted in Figures 5 (b) and (c). In Figure 5 (c), the points E and F are still rough, whereas in Figure 5 (b), the arc smoothly connects both sides, with A and B serving as optimal cutting points when the line segment's length is 1. Keeping the radius constant and altering the line segment's length adjusts the positions of points A and B. The specific process for determining the best cutting points is detailed in the pseudo-code of Algorithm 3. Thus, the arc of the chosen circle should align smoothly with the edge post-cutting, ensuring it only touches the edge tangentially without crossing it, to prevent creating additional sharp corners.

```
Algorithm 3 Path Smoothing ( )
Input: $T_{opt}$,w
Output: $T_{final}$
for $i = 2$ to $N_{opt} - 1$ do
    $l_1 = \| V_i, V_{i-1} \|_2, l_2 = \| V_i, V_{i+1} \|_2$;
    $\beta_1 =$ Angle $(V_i, V_{i-1}), \beta_2 =$ Angle $(V_i, V_{i+1})$;
    $l = \min(l_1, l_2)$/w;
    $A_x = V_{ix} + l \times \cos(\beta_1), A_y = V_{iy} + l \times \sin(\beta_1)$;
    $A'_x = V_{ix} + l \times \cos(\beta_2), A'_y = V_{iy} + l \times \sin(\beta_2)$
    r = $|l \times \tan((\beta_1 - \beta_2)/2)|$
    $C_x = A_x + r \times \cos(\beta_1 + \pi/2)$; $C_y = A_y + r \times \sin(\beta_1 + \pi/2)$;
    $C'_x = A_x + r \times \cos(\beta_1 - \pi/2)$; $C'_y = A_y + r \times \sin(\beta_1 - \pi/2)$;
    $C''_x = A'_x + r \times \cos(\beta_2 + \pi/2)$; $C''_y = A'_y + r \times \sin(\beta_2 + \pi/2)$;
    $C'''_x = A'_x + r \times \cos(\beta_2 - \pi/2)$; $C'''_y = A'_y + r \times \sin(\beta_2 - \pi/2)$;
    if $(C'_x = C''_x$ && $C'_y = C''_y) || (C'_x = C'''_x$ && $C'_y = C'''_y)$ then
        $C = C'$;
    else
        $C = C$;
    end if
    delete $V_i$;
    $(V_{i-1}, V_{i+1}) \leftarrow$ Insert $(A, AA', A')$;
end for
return $T_{final}$;
```

Fig. 4. Pseudo code Path Smoothing

1) Calculation of Optimal Radius: In Figure below, the triangle labled CBA and CBA' are identical in shape and size, making them congruent. The segment CA and CA', which represents the radius of a circle, are equal,and so are the internal angles of the triangle due to being congruent. Thus, the lengths of the sides AB and AB' are each half of the shortest side, through this proportion can be adjusted. To determine the radius for a smooth circular arc, its essential to calculate the angle formed between the connecting edges.

$$\alpha_i = \arctan\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right) - \arctan\left(\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}}\right)$$

$$\alpha_{inew} = 2\pi\alpha_i$$

To calculate the radius of a circle, one must first determine the angle between the vector from the circle's center to the node and the adjacent edge. At the i-th node, this angle is denoted by θ_i and is given by:

$$\beta_i = \frac{\alpha_i}{2}$$

where α is the angle between the two consecutive edges. The radius of the smoothing circle, r_i, can then be found using the tangent of the halved angle:

$$r_i = \tan(\beta_i)^* |AB|$$

where AB is the distance between two points on the edge.

After computing the radius, the center coordinates can be determined. Assuming the points A and A' represent the cuts, and given the radius r and angle θ, the central coordinates are calculated symmetrically around points A and A'. In total, four center coordinates are identified, which are illustrated in Figure 6, where the centers of circles C and C are coincident and represent the desired cutting circle.

The specific algorithm for this computation is detailed in the pseudo-code of Algorithm 3.

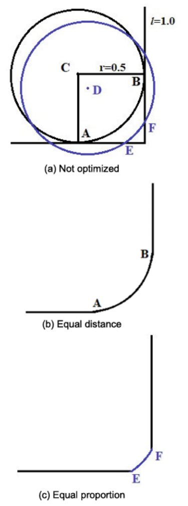


Fig. 5. Circles cut the edges formed by sub-paths from different points

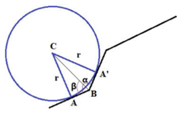


Fig. 6. Finding the radius of the circle

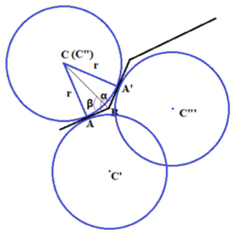


Fig. 7. Calculating the center coordinates

2) Path Cost Function equation: This analysis outlines the modifications in path cost as a result of employing a path smoothing approach. Initially, the cost is derived using the Euclidean formula between subsequent points $V_i(x_i, y_i)$ and $V_{i+1}(x_{i+1}, y_{i+1})$:

$$\|V_i, V_{i+1}\|_2 = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

wherein the sum of these distances defines both the initial and after-optimization path costs.

The smoothing strategy modifies all points within the set T_{opt}, excluding the start and end nodes, optimizing each point to enhance connectivity and smoothness. For example, focusing on node B from Figure 5, the strategy selects the midpoint of the segment l_{B1}, which is shorter than l_{B2}, as the optimal adjustment point.

The radius for the arc at node B, denoted as r_B, is calculated by:

$$r_B = \frac{1}{2}\tan(\beta) \times l_{B1}$$

where β indicates the angle adjustment needed at node B. The length of the arc, AA', subsequently becomes:

$$AA' = \frac{\pi - \alpha}{180} \times \pi \times r_B$$

The reduction in path cost due to this single adjustment is expressed as:

$$\triangle = l_{B1} - \frac{(\pi - \alpha) \times \pi \times r_B}{180}$$

Extending this approach across the path leads to a comprehensive calculation of the total path cost reduction achieved through the implemented smoothing strategy.

III. Simulation Results

Simulation are carried out to prove the effectiveness of the proposed algorithm. In the initial experiment, the effectiveness of the proposed algorithm is evaluated over various iterations on a consistent map layout. A subsequent study compares the performance of several algorithms, including RRT, Quick-RRT*, Fast-RRT, Informed RRT-Connect, and CAF-RRT*, across both maze and cluttered map environments to emphasize their respective strengths. Ultimately, the CAF-RRT* algorithm undergoes multiple tests in a complex maze environment to assess its adaptability and robustness.

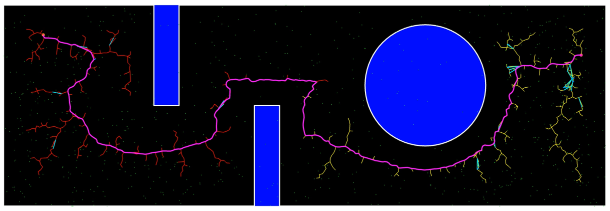


Fig. 8. Maze with turtlebot 3 , $dist_{unit} = 0.25, \triangle e = 10, p = 0.01, w = 5$

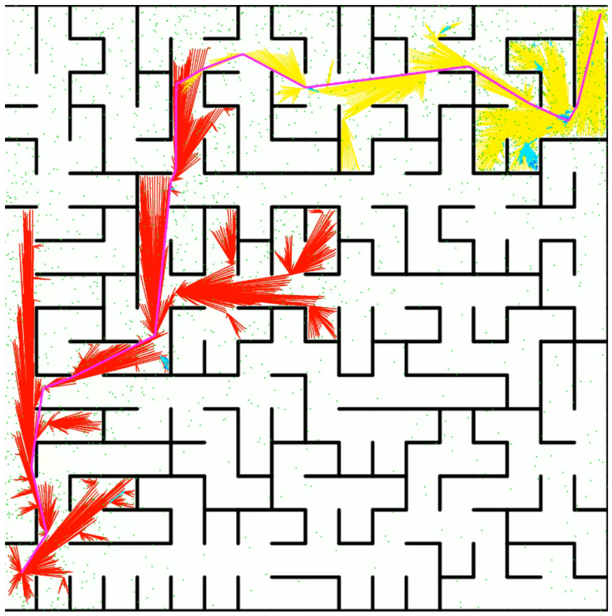
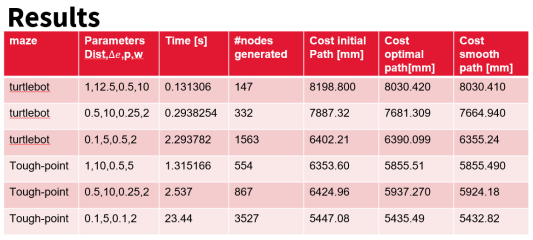


Fig. 9. Adaptability Experiment in Maze tough. $dist_{unit} = 1, \triangle e = 5, p = 0.1, w = 2$



Results

maze	Parameters Dist,Δe,p,w	Time [s]	#nodes generated	Cost initial Path [mm]	Cost optimal path[mm]	Cost smooth path [mm]
turtlebot	1,12.5,0.5,10	0.131306	147	8198.800	8030.420	8030.410
turtlebot	0.5,10,0.25,2	0.2938254	332	7887.32	7681.309	7664.940
turtlebot	0.1,5,0.5,2	2.293782	1563	6402.21	6390.099	6355.24
Tough-point	1,10,0.5,5	1.315166	554	6353.60	5855.51	5855.490
Tough-point	0.5,10,0.25,2	2.537	867	6424.96	5937.270	5924.18
Tough-point	0.1,5,0.1,2	23.44	3527	5447.08	5435.49	5432.82

Fig. 10. Results

A. Related Parameter settings

This section delves into the configuration of essential parameters within the proposed algorithm and examines how adjustments to these parameters affect path planning performance. the key parameters discussed include the optimization strategy's depth, distance and ration, the placement and the expansion radius used.

Variations in the distance and proportion parameters of the path optimization strategy show comparable influences. Larger parameter values lead to a more aggressive algorithm that requires fewer calls to the Collisionfree function, potentially diminishing the strategy's ability to lower path costs. On the other hand, smaller values increase the frequency of the Collisionfree procedure, improving the effectiveness in reducing path costs. Experiments to gauge these parameters' effects on algorithm performance across various maps revealed significant insights, with findings from a particular map detailed.

B. Performance Proposed Methods

The section on "Proposed Methods" introduces two distinct strategies for optimizing path planning: the *Optimize Equal Distance* method and the *Optimize Equal Proportion* method. In the Optimize Equal Distance approach, waypoints are evenly spaced using a parameter $\triangle e$ to minimize deviations in the path. In contrast, the Optimize Equal Proportion method adjusts distances proportionally based on a parameter p, tailoring the spacing to the specific characteristics of the path. Safety remains paramount in both strategies, with routine collision checks conducted to ensure the integrity of the path. The optimization efforts lead to significant cost reductions, achieving between 3-8 percent savings, whereas the smoothing technique contributes to a less pronounced cost reduction, generally less than 1 percent.

C. Unusual Results

Parameters that fall outside their acceptable ranges may lead to undesirable trajectories. This is illustrated with specific settings for parameters such as `dist_unit`, $\triangle e$, `proportion` p, and `w` for smoothing. The graph displayed depicts three distinct paths: the *Initial Path*, *Optimal Path*, and *Smoothed Path*. These are plotted with the x-position in millimeters on the horizontal axis and an unspecified physical measurement on the vertical axis.

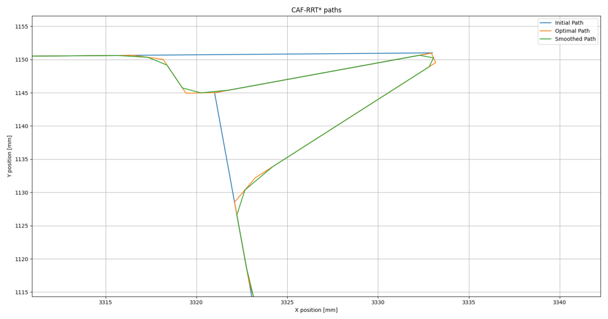


Fig. 11. Unwanted trajectories when $dist_{unit} >= 0.35$ and $\triangle e = 15$

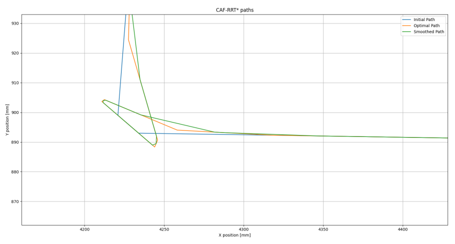


Fig. 12. Unwanted trajectories, $\triangle e$ greater than 20

The *Initial Path* begins with a lower value, rising smoothly as it progresses. The *Optimal Path* shows a similar trajectory but ascends more steeply towards its conclusion. In contrast, the *Smoothed Path* starts akin to the Optimal Path but transitions to a gentler, more even ascent. This graphical representation aids in evaluating the effects of varying path optimization strategies on a physical system.

IV. Conclusion

This paper evaluated two path optimization approaches—*Optimize Equal Distance* and *Optimize Equal Proportion*—which significantly improve autonomous path planning. The methods systematically reduce path deviations and adapt to environmental constraints, achieving a 3-8% reduction in path costs, with additional smoothing contributing marginally. While proving effective in ensuring safety, these strategies suggest potential enhancements when integrated with real-time processing and AI. Future research will explore their extension to 3D navigation and multi-agent systems to enhance their utility in dynamic settings.

References

[1] WANG, B., JU1, D., XU, F., FENG, C., & XUN, G. (2022, December). CAF-RRT∗: A 2D Path Planning Algorithm Based on Circular Arc Fillet Method. IEEE SYSTEMS, MAN AND CYBERNETICS SOCIETY SECTION, 127168-127181. Retrieved from https://ieeexplore.ieee.org/document/9969595

[2] Jeong, I.-B., Lee, S.-J., & Kim, J.-H. (2019, January). Quick-RRT∗: Triangular inequality-based implementation of RRT∗ with improved initial solution and convergence rate. Expert Systems with Applications, 82-90. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0957417419300326

[3] LaValle, S. M. (2006). Planning Algorithms . Cambridge University Press. Retrieved from https://msl.cs.uiuc.edu/planning/bookbig.pdf

[4] Miller, R. (1992). JOINING TWO LINES WITH A CIRCULAR ARC FILLET. Graphics Gems III (IBM Version), 193-198. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/B9780080507552500440