Fuzzy Adaptive RRT*N path planning and control of Autonomous Vehicle in CARLA ENPM661 Group 22

Abubakar Siddiq
M.Eng Robotics
UID: 120403422
UMD College Park
Email: absiddiq@umd.edu

Gayatri Davuluri
M.Eng Robotics
UID: 120304866
UMD College Park
Email: gayatrid@umd.edu

Dhana Santhosh reddy
M.Eng Robotics
UID: 120405570
UMD College Park
Email: js5162@umd.edu

***Abstract*—This report presents an implementation and evaluation of the Fuzzy Adaptive RRT*N path planning algorithm for autonomous vehicles in the CARLA simulator. The report is based on the Intelligent Adaptive RRT* algorithm proposed in the paper "Intelligent Adaptive RRT* Path Planning Algorithm for Mobile Robots" by Omar et al. We replicated the FA-RRT* algorithm and implemented it with autonomous vehicles navigating environments (Towns) in CARLA. The main goal of the Fuzzy Adaptive RRT*N is to improve the efficiency of path generation in terms of both computation time and path quality. It achieves this by incorporating fuzzy logic to adapt the RRT* algorithm parameters dynamically based on the local environment characteristics. Specifically, it adjusts the step size and goal biasing parameters using fuzzy rules that consider obstacle density and distance to the goal. In our implementation, we developed the algorithm from scratch in Python and integrated it with the CARLA client to control the autonomous vehicle. We evaluated its performance by conducting multiple runs with the same conditions and compared the results of RRT* and FA-RRT*N. Results show that the FA-RRT*N approach generates high-quality, collision-free paths while reducing computation times compared to the original RRT* algorithm.**

***Keywords*—Path Planning, RRT*, RRT*N, Fuzzy Logic, FA-RRT*N, CARLA.**

I. Introduction

In the rapidly evolving domain of robotics, the capability to navigate complex environments efficiently and effectively remains a cornerstone for advancing robot applications across various sectors. Path planning, in particular, plays a critical role in ensuring that robots can execute tasks with optimal efficiency and minimal human intervention. Among the myriad of strategies developed to enhance path planning, the RRT* (Rapidly-exploring Random Tree Star) algorithm has emerged as a significant breakthrough due to its flexibility and cost-effectiveness in navigating through intricate spaces.

This report delves into the practical implementation of a novel path planning algorithm, the Fuzzy Adaptive RRT**N (FA-RRT**N), which integrates fuzzy logic and normal probability distribution strategies into the traditional RRT* framework. The FA-RRT*N aims to optimize the path planning process by intelligently guiding the distribution of nodes, thus reducing computational overhead and enhancing path optimization.

To validate the effectiveness and real-world applicability of the FA-RRT*N algorithm, a comprehensive simulation was conducted using the CARLA Simulator. CARLA provides a robust, open-source platform for the development, training, and validation of autonomous driving systems in a realistic urban environment. This simulation aimed to replicate the conditions and challenges described in the theoretical model and to observe the behavior of the FA-RRT*N algorithm under dynamic conditions.

This report will cover the methodologies employed, the simulation setup in CARLA, the results observed, and a discussion on the implications of these results for future applications in robotic navigation. The successful implementation of the FA-RRT*N algorithm in CARLA not only demonstrates its potential in simulated urban environments but also sets the groundwork for further research and application in real-world scenarios.

II. Literature review

The proposed FA-RRT*N algorithm, as detailed by Omar et al. (2023), presents a notable advancement in sampling-based path planning for mobile robots. By integrating an intelligent fuzzy logic system, FA-RRT*N dynamically adjusts node generation based on environmental factors, improving adaptability and efficiency compared to traditional approaches. Building upon the foundation laid by RRT* and its variants, FA-RRT*N offers a compelling solution to the challenges of manual parameter tuning, demonstrating significant improvements in path optimization and computational efficiency. This innovation underscores the ongoing evolution of path planning algorithms, with FA-RRT*N representing a noteworthy contribution to the field of robotic navigation.

III. Methodology

This section delineates the methodologies employed in the development and execution of the Rapidly-exploring Random Trees Star (RRT*), RRT* Normal (RRT*N), and Fuzzy Adaptive RRT Normal (FA-RRT*N) algorithms. The incorporation of a controller aims to enhance the responsiveness and precision of path planning in dynamically changing environments.

*A. RRT**

The RRT* (Rapidly-exploring Random Trees Star) algorithm enhances the basic RRT by optimizing path efficiency and cost. Below is a summary of the key execution steps:

Step 1: Initialization and Node Generation

- **Node Generation:** The RRT* algorithm begins by initializing the start node (q_{start}) and generating random nodes (q_{rand}) within the configuration space Q, adhering to a maximum distance ($d_{\max}$) to ensure extensive coverage.
- **Node Placement:** Nodes are strategically placed to avoid obstacles, maintaining compliance with environmental constraints.

Step 2: Collision Checking

- **Obstacle Check:** Each node (q_{rand}) is verified to ensure it does not reside within an obstacle (q_{obs}), preventing the development of impractical paths.
- **Path Segment Validation:** The algorithm checks that no path segment from the nearest node to q_{rand} intersects with obstacles, ensuring path viability.

Step 3: Path Optimization

- **Cost Evaluation:** Unlike basic RRT, RRT* evaluates the cost of connecting q_{rand} to the tree, considering multiple nodes within a certain radius to find the most cost-effective connection.
- **Optimal Connection:** Selects the candidate node that offers the shortest and least costly path without obstacle interference, optimizing the path efficiency.

Step 4: Rewiring

- **Tree Optimization:** RRT* dynamically adjusts the tree by exploring whether existing nodes could benefit from rerouting through q_{rand}. Adjustments are made if a more cost-effective path is found.

Step 5: Iteration Until Goal is Reached

- **Continual Growth:** Repeats the process of node generation, checking, and rewiring iteratively until a satisfactory path to the goal (q_{goal}) is achieved or until resources are depleted.

*B. RRT*N*

The Rapidly-exploring Random Trees Normal (RRT*N) algorithm enhances the basic RRT* framework by strategically generating nodes using a normal distribution along a hypothetical line between the start (q_{start}) and goal (q_{goal}). RRT*N incorporates steps for node generation, collision checking, path optimization, and rewiring, all iteratively refined until a satisfactory path is established or computational limits are reached.

Algorithm 1 Generic RRT* Algorithm

```
1:  Initialize tree $T$ with node $q_{\text{start}}$
2:  for $k = 1$ to MAX_ITERATIONS do
3:      $q_{\text{rand}} \leftarrow$ GenerateRandomNode()
4:      if IsInObstacle($q_{\text{rand}}, q_{\text{obs}}$) then
5:          continue
6:      end if
7:      $q_{\text{nearest}} \leftarrow$ NearestNode($T, q_{\text{rand}}$)
8:      $q_{\text{new}} \leftarrow$ Steer($q_{\text{nearest}}, q_{\text{rand}}, d_{\max}$)
9:      if NotInCollision($q_{\text{new}}, q_{\text{nearest}}, q_{\text{obs}}$) then
10:         $Q_{\text{near}} \leftarrow$ FindNearNodes($T, q_{\text{new}}$, radius)
11:         $q_{\text{min}} \leftarrow q_{\text{nearest}}$
12:         $c_{\text{min}} \leftarrow$ Cost($q_{\text{nearest}}$) + Distance($q_{\text{nearest}}, q_{\text{new}}$)
13:         for each $q_{\text{near}}$ in $Q_{\text{near}}$ do
14:             if Cost($q_{\text{near}}$) + Distance($q_{\text{near}}, q_{\text{new}}$) $< c_{\text{min}}$ and
                NotInCollision($q_{\text{near}}, q_{\text{new}}, q_{\text{obs}}$) then
15:                 $q_{\text{min}} \leftarrow q_{\text{near}}$
16:                 $c_{\text{min}} \leftarrow$ Cost($q_{\text{near}}$) + Distance($q_{\text{near}}, q_{\text{new}}$)
17:             end if
18:         end for
19:         AddNode($T, q_{\text{new}}, q_{\text{min}}$)
20:         Rewire($T, Q_{\text{near}}, q_{\text{new}}, q_{\text{obs}}$)
21:     end if
22:     if Distance($q_{\text{new}}, q_{\text{goal}}$) $\leq$ GOAL_THRESHOLD then
23:         return PathToStart($q_{\text{new}}$)
24:     end if
25: end for
26: return FAILURE =0
```

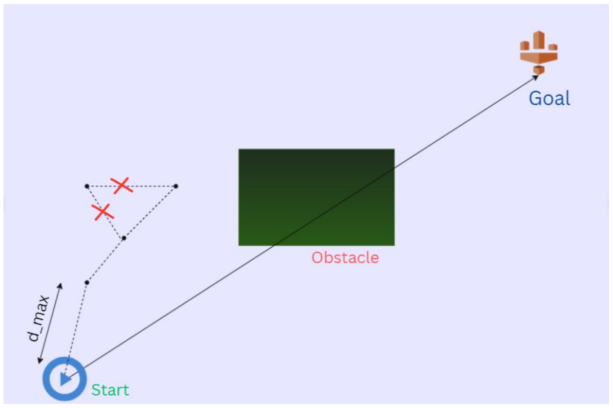


Fig. 1. RRT*N approach to node generation

Step 1: Initialization and Node Generation

- **Node Generation:** RRT*N initializes with the start node q_{start}. Unlike RRT/RRT*, nodes q_{rand} are generated along a normal distribution centered on a hypothetical straight line between q_{start} and the goal q_{goal}. This optimizes search efficiency.
- **Guided Distribution:** Standard deviation of the distribution adapts based on obstacle density and distance to the goal, further focusing the search area.

Step 2: Collision Checking

- **Obstacle Verification:** Generated nodes q_{rand} are checked to ensure they don't reside within obstacles q_{obs}.
- **Path Segment Validation:** The line segment connecting q_{rand} to the nearest existing node is checked for obstacle collisions to maintain path feasibility.

Step 3: Path Optimization

- **Cost Evaluation and Connection:** RRT*N evaluates the cost of connecting q_{rand} to multiple nearby nodes, not just the nearest one. It selects the connection offering the shortest, obstacle-free path.
- **Efficiency Optimization:** The normal distribution influences this process, biasing node generation towards the potential optimal path, leading to more efficient solutions.

Step 4: Rewiring

- **Dynamic Tree Adjustment:** RRT*N identifies if existing nodes would benefit from rerouting their paths through q_{rand}. If a lower-cost path exists, the tree is adjusted accordingly.

Step 5: Iteration and Convergence

- **Iterative Refinement:** These steps repeat until a satisfactory path to the goal (q_{goal}) is found or computational limits are reached.
- **Focused Exploration:** RRT*N's guided node placement often leads to faster convergence towards an optimal solution compared to RRT.

C. *FA-RRT*N*

The FA-RRT*N algorithm addresses a key limitation of RRT*N by automatically adapting to different environments through fuzzy adaptive techniques. Instead of manually setting the standard deviation, FA-RRT*N dynamically adjusts it based on the characteristics of the environment.

Algorithm 2 RRT*N Algorithm

```
Require: Start position $q_{\text{start}}$, Goal position $q_{\text{goal}}$, Obstacle space $q_{\text{obs}}$
Ensure: Path from $q_{\text{start}}$ to $q_{\text{goal}}$
1: Initialize tree $T$ with node $q_{\text{start}}$
2: for $k = 1$ to MAX_ITERATIONS do
3:     $q_{\text{rand}} \leftarrow$ GenerateNodeUsingNormalDistribution($q_{\text{start}}$, $q_{\text{goal}}$)
4:     if IsInObstacle($q_{\text{rand}}$, $q_{\text{obs}}$) then
5:         continue
6:     end if
7:     $q_{\text{nearest}} \leftarrow$ NearestNode($T$, $q_{\text{rand}}$)
8:     $q_{\text{new}} \leftarrow$ Steer($q_{\text{nearest}}$, $q_{\text{rand}}$)
9:     if NotInCollision($q_{\text{new}}$, $q_{\text{nearest}}$, $q_{\text{obs}}$) then
10:        $Q_{\text{near}} \leftarrow$ FindNearNodes($T$, $q_{\text{new}}$, predefined_radius)

11:        $q_{\text{min}} \leftarrow q_{\text{nearest}}$
12:        $c_{\text{min}} \leftarrow$ Cost($q_{\text{nearest}}$) + Distance($q_{\text{nearest}}$, $q_{\text{new}}$)
13:        for all $q_{\text{near}}$ in $Q_{\text{near}}$ do
14:            if Cost($q_{\text{near}}$) + Distance($q_{\text{near}}$, $q_{\text{new}}$) $< c_{\text{min}}$ and NotInCollision($q_{\text{near}}$, $q_{\text{new}}$, $q_{\text{obs}}$) then
15:                $q_{\text{min}} \leftarrow q_{\text{near}}$
16:                $c_{\text{min}} \leftarrow$ Cost($q_{\text{near}}$) + Distance($q_{\text{near}}$, $q_{\text{new}}$)
17:            end if
18:        end for
19:        AddNode($T$, $q_{\text{new}}$, $q_{\text{min}}$)
20:        Rewire($T$, $Q_{\text{near}}$, $q_{\text{new}}$, $q_{\text{obs}}$)
21:    end if
22:    if Distance($q_{\text{new}}$, $q_{\text{goal}}$) $\leq$ GOAL_THRESHOLD then
23:        return PathToStart($q_{\text{new}}$)
24:    end if
25: end for
26: return FAILURE =0
```

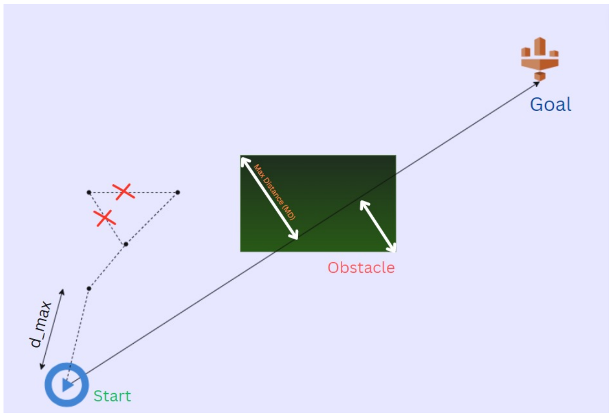


Fig. 2. FA-RRT*N approach to node generation

Initially, a normal line is established from the start to the goal as a reference, and nodes are generated around this line using the normal distribution formula. The algorithm then automatically sets the standard deviation based on the largest obstacle along this straight line path. The maximum distance from the line to the edges of the obstacle is measured on both sides, and the larger of the two distances serves as input to the Fuzzy Inference System (FIS).

Using a Mamdani FIS with minimum implication and maximum aggregation, the system takes the largest maximum distance (MD) as input and outputs a standard deviation coefficient (SD). Both input and output are defined with five membership functions: very small (VS), small (S), medium (M), large (L), and very large (VL). Generalized bell membership functions are used for inputs, defined as follows.

$$\mu_{MD_i}(x) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}} \quad (1)$$

Where a is the half width, c is the center of the corresponding membership function, and b changes the slope at the crossover points

The fuzzy rules set for the FIS includes five rules, each associating a range of MD distances with a corresponding SD. These rules govern the mapping from input MD to output SD.

- **Rule 1**: If the MD distance is very small, then the SD is very small.

Algorithm 3 Fuzzy Adaptive RRT*N Algorithm

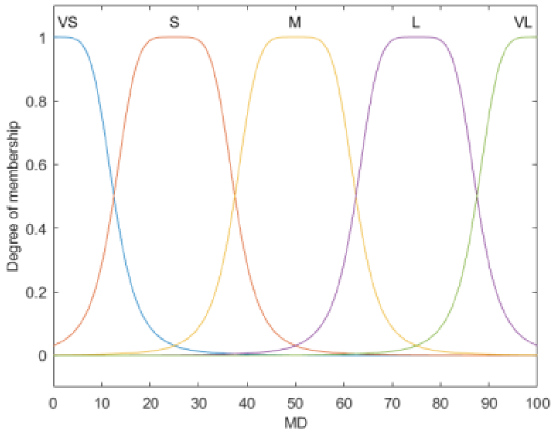
```
Require: Start position $q_{\text{start}}$, Goal position $q_{\text{goal}}$, Map dimensions $map\_width, map\_height$, Obstacle clearance $clearance$
Ensure: Path from $q_{\text{start}}$ to $q_{\text{goal}}$
 1: Import necessary libraries (numpy, cv2, matplotlib, skfuzzy, math, time)
 2: Set map dimensions ($map\_width = 6000$, $map\_height = 2000$)
 3: Initialize the fuzzy logic controller with rules for node generation and steering
 4: $obstacle\_map \leftarrow$ CreateMap($map\_width, map\_height, clea$
 5: Initialize tree $T$ with node $q_{\text{start}}$
 6: for $k = 1$ to MAX_ITERATIONS do
 7:     $q_{\text{rand}} \leftarrow$ GenerateNodeFuzzily($q_{\text{start}}$, $q_{\text{goal}}$) {Using fuzzy logic for diversified node generation}
 8:     if IsInObstacle($q_{\text{rand}}$, $obstacle\_map$) then
 9:         continue
10:     end if
11:     $q_{\text{nearest}} \leftarrow$ NearestNode($T$, $q_{\text{rand}}$)
12:     $q_{\text{new}} \leftarrow$ SteerFuzzily($q_{\text{nearest}}$, $q_{\text{rand}}$) {Using fuzzy control for smooth and adaptable steering}
13:     if NotInCollision($q_{\text{new}}$, $q_{\text{nearest}}$, $obstacle\_map$) then
14:         $Q_{\text{near}} \leftarrow$ FindNearNodes($T$, $q_{\text{new}}$)
15:         $q_{\text{min}} \leftarrow q_{\text{nearest}}$
16:         $c_{\text{min}} \leftarrow$ Cost($q_{\text{nearest}}$) + Distance($q_{\text{nearest}}$, $q_{\text{new}}$)
17:         for all $q_{\text{near}}$ in $Q_{\text{near}}$ do
18:             if Cost($q_{\text{near}}$) + Distance($q_{\text{near}}$, $q_{\text{new}}$) ¡ $c_{\text{min}}$ and NotInCollision($q_{\text{near}}$, $q_{\text{new}}$, $obstacle\_map$) then
19:                 $q_{\text{min}} \leftarrow q_{\text{near}}$
20:                 $c_{\text{min}} \leftarrow$ Cost($q_{\text{near}}$) + Distance($q_{\text{near}}$, $q_{\text{new}}$)
21:             end if
22:         end for
23:         AddNode($T$, $q_{\text{new}}$, $q_{\text{min}}$)
24:         Rewire($T$, $Q_{\text{near}}$, $q_{\text{new}}$, $obstacle\_map$)
25:     end if
26:     if Distance($q_{\text{new}}$, $q_{\text{goal}}$) $\leq$ GOAL_THRESHOLD then
27:         return PathToStart($q_{\text{new}}$)
28:     end if
29: end for
30: return FAILURE =0
```

- **Rule 2**: If the MD is small, then the SD is small.
- **Rule 3**: If the MD is medium, then the SD is medium.
- **Rule 4**: If the MD is large, then the SD is large.
- **Rule 5**: If the MD is very large, then the SD is very large.

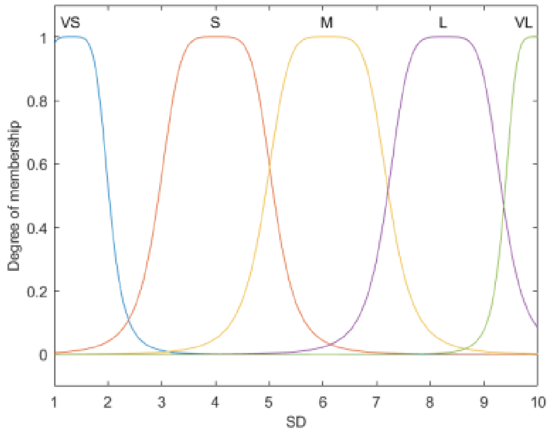
The membership functions for both input and output are evaluated using the minimum implication operator ($\wedge$).

$$\lambda_i = \mu_{SD_i}(MD) \tag{2}$$

$$\mu_{SD'_i} = \lambda_i \wedge \mu_{SD_i} \tag{3}$$



(a)



(b)

Fig. 3. (a) Fuzzy system input membership functions (b) Fuzzy system output membership functions

Then, the fuzzy sets of each rule are combined using the max operator. Finally, the centroid method is employed for defuzzification to determine the appropriate standard deviation.

$$\mu_{SD'} = \bigcup_{i=1}^{r} \mu_{SD'_i} \tag{4}$$

$$SD^* = \frac{\sum \mu_{SD'}(SD) \cdot SD}{\sum \mu_{SD'}(SD)} \tag{5}$$

The output of the FIS, representing the adjusted standard deviation, is integrated back into the algorithm to improve path planning efficiency. A flowchart summarizing the FA-RRT*N algorithm illustrates its operation and integration of fuzzy adaptive techniques.

IV. Results

After the algorithm is developed we have tested the both generic RRT* and the FA-RRT*N algorithm on multiple runs (25) by keeping the same start and end goal and evaluated the performance of both of them by comparing, *the number of*

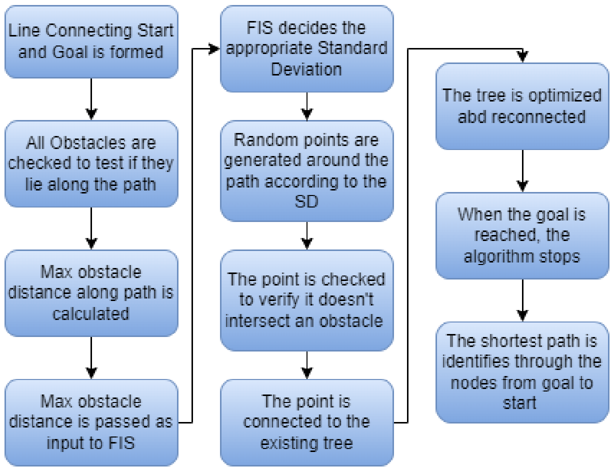
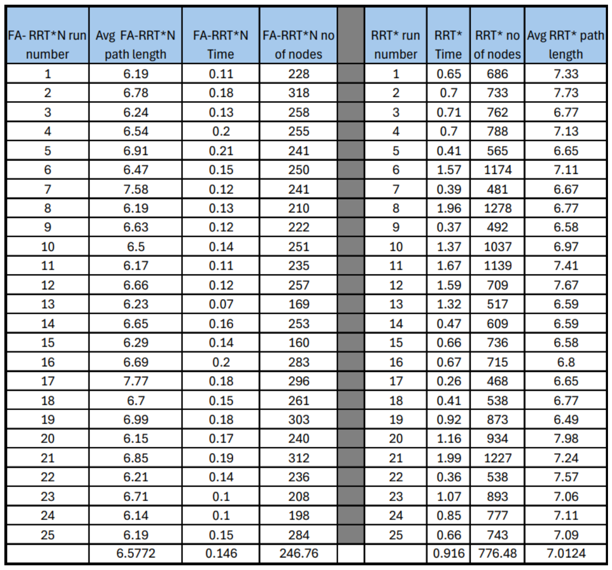

Line Connecting Start and Goal is formed
All Obstacles are checked to test if they lie along the path
Max obstacle distance along path is calculated
Max obstacle distance is passed as input to FIS
FIS decides the appropriate Standard Deviation
Random points are generated around the path according to the SD
The point is checked to verify it doesn't intersect an obstacle
The point is connected to the existing tree
The tree is optimized abd reconnected
When the goal is reached, the algorithm stops
The shortest path is identifies through the nodes from goal to start

Fig. 4. Fuzzy Adaptive-RRT*N Algorithm Flow chart

nodes generated, path length and the time taken to generate the path, all of it are discussed further in the subsections.



FA- RRT*N run number	Avg FA-RRT*N path length	FA-RRT*N Time	FA-RRT*N no of nodes		RRT* run number	RRT* Time	RRT* no of nodes	Avg RRT* path length
1	6.19	0.11	228		1	0.65	686	7.33
2	6.78	0.18	318		2	0.7	733	7.73
3	6.24	0.13	258		3	0.71	762	6.77
4	6.54	0.2	255		4	0.7	788	7.13
5	6.91	0.21	241		5	0.41	565	6.65
6	6.47	0.15	250		6	1.57	1174	7.11
7	7.58	0.12	241		7	0.39	481	6.67
8	6.19	0.13	210		8	1.96	1278	6.77
9	6.63	0.12	222		9	0.37	492	6.58
10	6.5	0.14	251		10	1.37	1037	6.97
11	6.17	0.11	235		11	1.67	1139	7.41
12	6.66	0.12	257		12	1.59	709	7.67
13	6.23	0.07	169		13	1.32	517	6.59
14	6.65	0.16	253		14	0.47	609	6.59
15	6.29	0.14	160		15	0.66	736	6.58
16	6.69	0.2	283		16	0.67	715	6.8
17	7.77	0.18	296		17	0.26	468	6.65
18	6.7	0.15	261		18	0.41	538	6.77
19	6.99	0.18	303		19	0.92	873	6.49
20	6.15	0.17	240		20	1.16	934	7.98
21	6.85	0.19	312		21	1.99	1227	7.24
22	6.21	0.14	236		22	0.36	538	7.57
23	6.71	0.1	208		23	1.07	893	7.06
24	6.14	0.1	198		24	0.85	777	7.11
25	6.19	0.15	284		25	0.66	743	7.09
	6.5772	0.146	246.76			0.916	776.48	7.0124

Fig. 5. Data collected over the 25 runs

The example output of the RRT* is shown in fig 6:

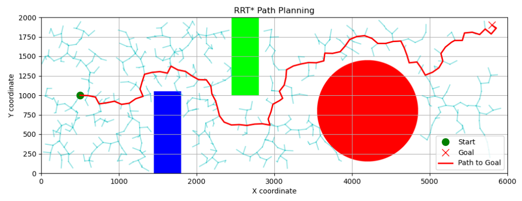


Fig. 6. RRT* Path planning results

The example output of the FA-RRT*N is shown in fig 7:

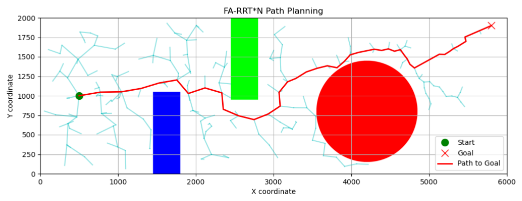


Fig. 7. FA-RRT*N Path planning results

A. *Algorithm performance evaluation*

Data is collected for the 25 runs of both the algorithms for the same start and end goal, when compared the data, we can clearly observe the difference between these two algorithms. The plots are shown in the below figures. *(orange is for RRT* and blue is for FA-RRT*N)*

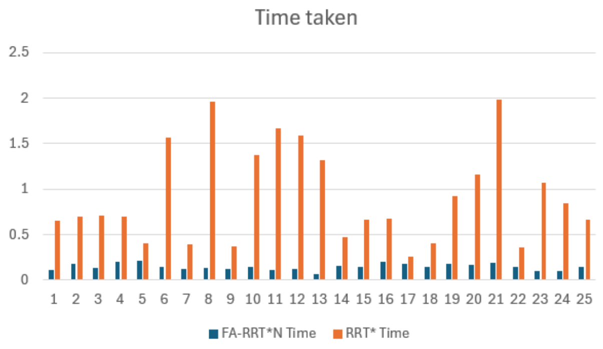


Fig. 8. Time taken to generate over the runs

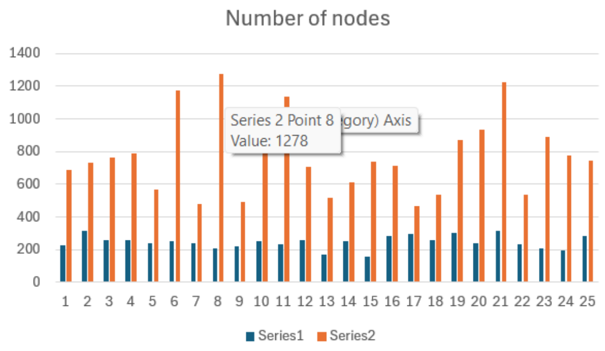


Fig. 9. Number of nodes generated over the runs

The average time taken to generate a path for RRT* is 0.9156s where as FA-RRT*N took only **0.146s**. And average number of nodes taken to generate a path for RRT* is 776 where as FA-RRT*N took only **247**. Finally, the average length of the path generated for this particular scenario for RRT* is 7.0124m where as for FA-RRT*N it is only only **6.5772m**.

From this we can clearly say that the FA-RRT*N algorithm performs extraordinarily well when compared to the RRT*. FA-RRT*N will take only 15.94 % of the time taken by RRT*

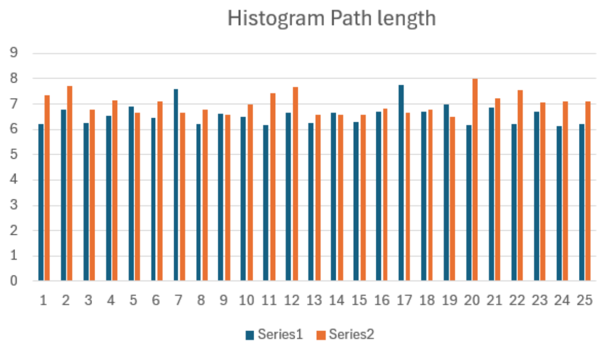


Fig. 10. Path length over the runs

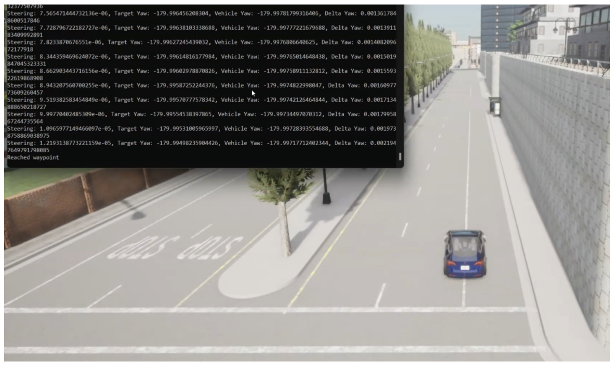
and will only generate **31.77%** of the nodes generated by RRT*. And will generate a shorter path compared to RRT* **74%** of the time.

B. *Simulation Results - CARLA*

Once the algorithm is developed and tested on 2D binary maps to generate a path from start goal to end goal in static obstacle environments we have moved on to simulating the results. The code outputs a set of way-points for the path and we utilized this to simulate a vehicle in CARLA. CARLA (CAR Learning to Act) is an open-source autonomous vehicle simulator that consists of multiple real life vehicle models and towns created using unreal engine, it is one of the most realistic simulators used widely in the autonomous vehicle industry.

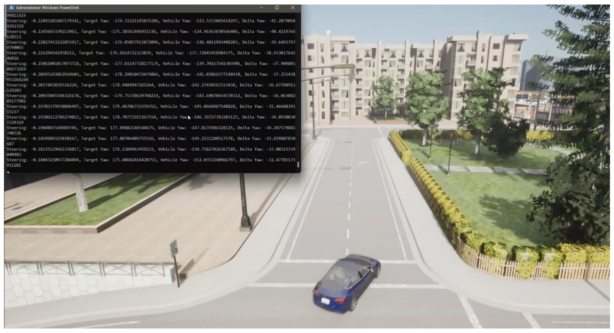
We first developed a script that will take these way-points as inputs and will steer the vehicle (tesla model 3 in this case) along these way-points in the town selected. It is a must that the CARLA engine(server) to be started and running up before executing the script. One of the issues we faced during testing this is, CARLA is a really heavy software that will utilize a huge amount of GPU while running and sometimes the server that is initiated for a previous run will stay in the cache and will prevent the new script (client) from connecting to a new server instance that we initiated, so before running the new instance it is really important to end the task of the previous instance from task manager, this will make sure that only one server is up and running at any instant.
The *drive to way-point* function is the core component of the script that navigates the vehicle to each way-point. It calculates the distance between the vehicle's current position and the target way-point. If the distance is less than 2 units, it considers the way-point reached and moves to the next one. Otherwise, it computes the desired steering angle by finding the difference between the vehicle's current heading and the direction toward the way-point. The steering angle is then normalized and applied to the vehicle's control, along with a constant throttle value (0.5) to move the vehicle forward. The function also updates the camera view to follow the vehicle's movement. The script iterates through the list of way-points, invoking *drive to way-point* for each one, and pauses briefly after reaching each way-point for clarity.

The simulation video is included in the presentation and can be accessed by the link attached here. Snip shots of the simulation are in the below pictures.



```
32377507936
Steering: 7.565471444732136e-06, Target Yaw: -179.996456208304, Vehicle Yaw: -179.99781799316406, Delta Yaw: 0.001361784
8600517846
Steering: 7.728796722182727e-06, Target Yaw: -179.99638103338688, Vehicle Yaw: -179.99777221679688, Delta Yaw: 0.0013911
83409992891
Steering: 7.8233870676551e-06, Target Yaw: -179.99627245439032, Vehicle Yaw: -179.9976806640625, Delta Yaw: 0.0014082096
72177918
Steering: 8.344359469624072e-06, Target Yaw: -179.99614816177984, Vehicle Yaw: -179.99765014648438, Delta Yaw: 0.0015019
847045323331
Steering: 8.662903443716156e-06, Target Yaw: -179.99602978870826, Vehicle Yaw: -179.99758911132812, Delta Yaw: 0.0015593
22619868908
Steering: 8.943207560700255e-06, Target Yaw: -179.99587252244376, Vehicle Yaw: -179.9974822998047, Delta Yaw: 0.00160977
73609260457
Steering: 9.519382583454849e-06, Target Yaw: -179.99570777578342, Vehicle Yaw: -179.99742126464844, Delta Yaw: 0.0017134
888650218727
Steering: 9.99770402485309e-06, Target Yaw: -179.99554538397865, Vehicle Yaw: -179.99734497070312, Delta Yaw: 0.00179958
67244735564
Steering: 1.0965977149466097e-05, Target Yaw: -179.99531005965997, Vehicle Yaw: -179.99728393554688, Delta Yaw: 0.001973
8758869038975
Steering: 1.2193138773221159e-05, Target Yaw: -179.99498235904426, Vehicle Yaw: -179.99717712402344, Delta Yaw: 0.002194
7649791798085
Reached waypoint
```

Fig. 11. CARLA simulation snip shot 1



```
99811429
Steering: -0.22893281607179541, Target Yaw: -174.72321145835286, Vehicle Yaw: -133.5153045654297, Delta Yaw: -41.2079068
9292318
Steering: -0.2245665370213961, Target Yaw: -175.38561496951536, Vehicle Yaw: -134.96363830566406, Delta Yaw: -40.4219766
638513
Steering: -0.22027433221055917, Target Yaw: -176.05057913872096, Vehicle Yaw: -136.4011993408203, Delta Yaw: -39.6493797
9790065
Steering: -0.216294542458112, Target Yaw: -176.6614722323039, Vehicle Yaw: -137.72845458984375, Delta Yaw: -38.933017642
46016
Steering: -0.21061001037073718, Target Yaw: -177.61147728177176, Vehicle Yaw: -139.70167541503906, Delta Yaw: -37.909801
86673269
Steering: -0.20695243862924603, Target Yaw: -178.29010472474866, Vehicle Yaw: -141.03866577148438, Delta Yaw: -37.251438
953264284
Steering: -0.2037443839516224, Target Yaw: -178.9484947265264, Vehicle Yaw: -142.27450561523438, Delta Yaw: -36.67398911
129203
Steering: -0.20035045584321678, Target Yaw: -179.75378639748214, Vehicle Yaw: -143.69070434570312, Delta Yaw: -36.063082
05177902
Steering: -0.19703379950846497, Target Yaw: 179.46706733359352, Vehicle Yaw: -145.0668487548828, Delta Yaw: -35.46608391
15237
Steering: -0.19386112786274023, Target Yaw: 178.70771915267554, Vehicle Yaw: -146.39727783203125, Delta Yaw: -34.8950030
1529324
Steering: -0.1904865548985596, Target Yaw: 177.89882148544675, Vehicle Yaw: -147.8135986328125, Delta Yaw: -34.287579881
740726
Steering: -0.1869989325038167, Target Yaw: 177.08706409755516, Vehicle Yaw: -149.2531280517578, Delta Yaw: -33.659807850
687
Steering: -0.18335129661336017, Target Yaw: 176.2384963459233, Vehicle Yaw: -150.75827026367188, Delta Yaw: -33.00323339
040483
Steering: -0.18043250977284894, Target Yaw: 175.48682414420753, Vehicle Yaw: -152.0353240966797, Delta Yaw: -32.47785175
911281
```

Fig. 12. CARLA simulation snip shot 2

V. Conclusion

In conclusion, we implemented and evaluated the Fuzzy Adaptive RRT*N (FA-RRT*N) path planning algorithm for autonomous vehicles in the CARLA simulator. FA-RRT*N incorporates fuzzy logic to dynamically adapt sampling parameters based on local obstacle density. Compared to the standard RRT* algorithm, FA-RRT*N demonstrated significant improvements in computational efficiency (**84%** reduction in time, **68%** fewer nodes explored) while generating shorter, higher-quality paths in most test cases. The successful simulation of autonomous vehicle navigation in CARLA using FA-RRT*N paths highlights the algorithm's effectiveness and real-world applicability. This adaptive sampling approach shows promise for enhancing path planning for robotic systems operating in complex environments.

References

[1] Khattab, O., Yasser, A., Jaradat, M. A., Romdhane, L. (2023). Intelligent Adaptive RRT* Path Planning Algorithm for Mobile Robots. 2023 Advances in Science and Engineering Technology International Conferences, ASET 2023. https://doi.org/10.1109/ASET56582.2023.10180740