RRT-A* Motion Planning Algorithm for Mobile Robot

Authors:

Rohith Vikram Saravanan (119474198)
rohithvs@umd.edu

Poojan Desai (119455760)
Pndesai9@umd.edu

Robotics, University of Maryland, College Park, United States

***Abstract*— The RRT is a powerful sampling-based path planning algorithm that is used to generate paths in complex and dense obstacle environments. The problem with RRT is that the path generated by it is not an optimal path. RRT generates and selects nodes randomly, which has varying total planning costs. To optimize the path generated by RRT, we are combining the A* star and RRT algorithms, which is the RRT-A* algorithm. The path generated by the RRT-A* algorithm is closer to the optimal path. Moreover, the RRT-A* algorithm is less time-consuming and more efficient in terms of computing the optimal path and planning costs.**

Keywords*— *RRT, Optimal path, A*, RRT-A*, planning cost

I. Introduction

Path planning is a method that is used to find the optimal path between a defined start point and a goal point while avoiding obstacles between them. The path planning method generally considers various constraints, such as geometric constraints, nonholonomic constraints, kinodynamic constraints, and dynamic constraints, while finding the optimal path between the start node and the goal node. It also takes into account the time taken, algorithmic optimality, and computational efficiency while finding the path. Path planning algorithms have been gaining popularity in recent times. Many path-planning algorithms are being implemented in the autonomous vehicle industry and the medical robotics industry, for example, in endoscopy. One of the earliest path planning algorithms was the geometric constraint path planning algorithm. The geometric constraint path planning computes the geometric environment containing obstacles and generates a geometric curve, which is used to connect the start position and the goal position[4]. The disadvantage of this algorithm is that it does not take into account kinematic and dynamic constraints such as time, velocity, acceleration, etc. To account for those constraints, non-holonomic differential constraints were introduced to the path planning algorithms. This takes into account the time, velocity, and acceleration constraints while computing the path from the start position to the goal position. So this eliminated the constraint of considering the robot as a point object.[2] The next advancement in the field of path planning was kinodynamic motion planning. The kinodynamic motion planning approach considers the dynamic of the robot or vehicle. It considers the balancing problems for the robot. Even though this approach considers the dynamic behavior of the robot, it fails to consider the dynamic behavior of the environment. To overcome this limitation, dynamic motion planning was introduced. Dynamic motion planning[2] not only

considers the dynamic constraints of the robot but also considers the dynamic behavior of the environment. The dynamic motion planning process considers the moving obstacles in the path planning environment. For example, in autonomous vehicles, dynamic motion planning is implemented to consider other vehicles' motion according to the planning environment.[2]

Sampling-based algorithms RRT is a powerful algorithm widely used in the field of robotics for pathfinding. RRT is specifically used to generate paths in highly complex and dimensional environments. RRT is effective in searching the complete configuration space and generating a feasible path between the start and goal positions[3].

Aside from the advantages of the RRT algorithm, the path generated by it is not optimal. To optimize the path generated by the RRT, an improved version of the RRT algorithm, the RRT* algorithm, was introduced. The RRT* algorithm computes the cost-to-come for each random node generated and does rewiring according to the cost-to-come for every node. This way, the RRT* algorithm generates a more optimal path when compared to the RRT algorithm. Moreover, the random tree generated by the RRT* algorithm is more symmetrical when compared to the RRT algorithm. Even though the path generated is optimal, it is not optimal up to expectations. To optimize the path generated, different variations of RRT are introduced. Out of the variations, the RRT-A* algorithm generates a more optimal path compared to RRT and RRT*[1].

II. Method

Functions and Class used:

1. A class Point is defined, which has the coord attribute to store the coordinates of a point, the cost attribute to store the cost to reach that point, and the parent attribute to store the parent node of that point. The class also has getters and setters to access these attributes.

2. Function Eucledian_Distance is defined, which takes two points and calculates the Eucledian Distance between those points.

3. Function steer is defined, which is used to find a point along the line joining the parent node and the randomly generated node. it is used to steer towards the random node according to the provided threshold.

4. Function LineCheck is defined, which is used to check whether a point lies on the line segment that is generated between the other two given points.

5. Function Intersection_Detection is used to check whether two lines generated by the 4 points given intersect.

6. Function Obstacle_Collision is used to determine whether the line segment defined by the 2 given points intersects with any of the obstacle points defined inside the obstacle space.

7. The functions User_Inputs_Start and User_Inputs_Goal are used to get the input values from the user for the start node and the goal node.

8. Finally, the main function is defined, which uses all the functions defined above and computes the RRT-A* algorithm as follows.

Step 1: The start node and the goal node are taken as user inputs from the user.

Step 2: Obstacles are created in the environment and they are stored in a list created Obs_Pts. By using the list Obs_Pts the obstacles are plotted in the path planning environment using matplotlib.

Step 3: An dictionary StepDistances is created and the distances between every generated node and its parent node are calculated and the calculated distance is stored in the StepDistances dictionary. is

Step 4: The number of iterations for generating the path tree is defined and for every iteration a random node is generated and the distance between the randomly generated node and every other node in the map is searched in the StepDistances dictionary. If the distance is not present in the dictionary then that distance is calculated. The obtained distance between the random node and every other node is then compared and the node with the least distance from the random node is selected.

Step 5: Then the steer function is called to move toward the direction from the closest node to the random node with the distance provided by the threshold.

Step 6: Then the line generated from the closest node and the random node is checked whether it colloids with any obstacle in the environment. If the line does not collide with any obstacle in the environment then the heuristic cost function of the A* algorithm is implemented into the RRT algorithm to optimize the path generated in the RRT algorithm.

TOTAL COST FUNCTION:

$$f(P) = g(P) + h(P)$$ [1]

where f(P) is the total cost of that node from the star position to the goal position. The g(P) is the cost-to-come of the node and h(P) is the heuristic function ie. the cost-to-go for that node. The Total cost function used in A* algorithm is implemented into the RRT to optimise the path generated by the RRT.[4]

Step 7: Some random nodes are generated within the threshold distance from the previously generated node and for each random node generated the cost-to-come from the start point is then the Euclidean distance between that point and the goal node is calculated. With the calculated cost-to-come and cost-to-go distances, the total cost for every node is calculated.

Step 8: The total cost calculated for every randomly generated node is compared and the random node with the least total cost is selected.

Step 9: Then the line is generated towards that random node with the least total cost and the previously generated node with the distance according to the step value.

Step 10: Then the parent node of the newly generated node is set to the previously generated node. Then the newly generated node is appended to the list of nodes.

Step 11: Then all the nodes generated in the list of nodes are checked if they have reached the threshold of the goal node.

Step 12: If any of the nodes has reached the goal threshold then the goal is reached. The goal point is appended to the list of nodes. The parent of the goal node is set to the node which is within the goal node threshold.

Step 13: Then the backtracking is done considering the goal node as the start node and the list of parent nodes as the next node and it continues till it reaches the start node.

Step 14: Then the forward tracking, as well as the backtracking, is plotted in the environment using the matplotlib.

III. RESULT

To determine which method generates paths with a better optimal path and greater efficiency, we examined the paths produced by the RRT and RRT-A* algorithms. To consider which algorithm performs well in a free space and a crowded space, we have considered two maps for both algorithms. The total cost and time are computed for all the paths generated.

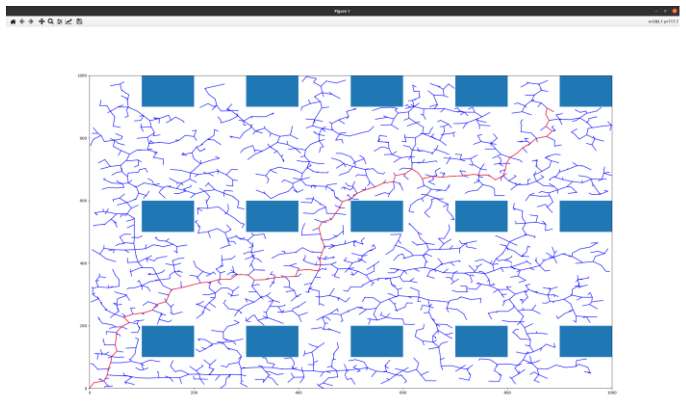

Fig 1: RRT path map for less dense environment

The above map represents the path map generated by RRT in a less dense environment. The RRT algorithm has explored the whole map with a path generated in all obstacle-free space. We can see that the path generation for RRT has a total cost of 1720 and the time taken for path generation is 11.19 seconds.

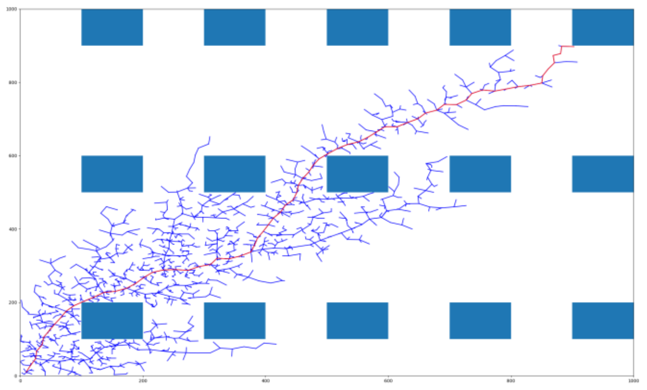

Fig 2: RRT-A* path map for less dense environment

The above map represents the path map generated by RRT-A* in a less dense environment. The RRT-A* algorithm generates nodes toward the goal rather than exploring the whole map. The path generation for RRT-A* has a total cost of 1373 and the time taken for path generation is 24.78 seconds.

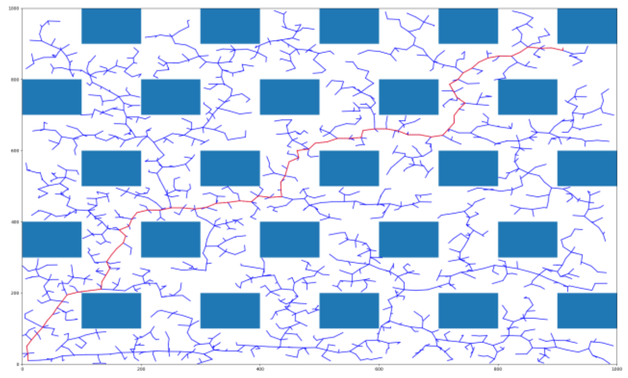

Fig 3: RRT path map for high dense environment

The above map represents the path map generated by RRT in a high dense environment. The path generation for RRT has a total cost of 1577 and the time taken for path generation is 4.02 seconds.

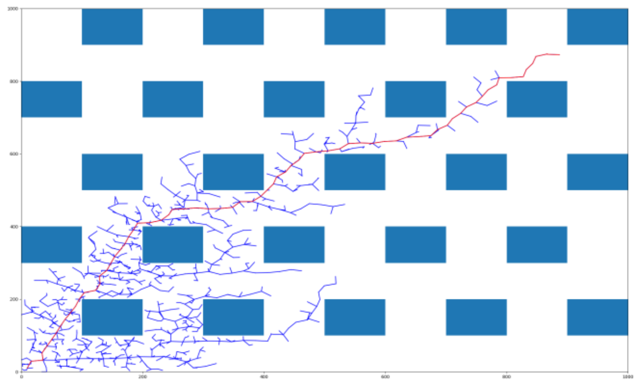

Fig 4: RRT-A* path map for high dense environment

The above map represents the path map generated by RRT-A* in a high dense environment. The path generation for RRT-A* has a total cost of 1376 and the time taken for path generation is 8.17 seconds.

Thus, we can infer that the total cost to generate the RRT-A* algorithm is less for both dense and less dense environments. Whereas the time required to calculate the RRT algorithm is less when compared to the RRT-A* algorithm.

IV. CONCLUSION

Thus, from the results obtained from the RRT and RRT-A* algorithms for the dense and less dense environments, we can infer that the RRT algorithm has a high total cost but consumes less time to generate paths. whereas the RRT-A* algorithm has a lower total cost and higher time consumption when compared to the RRT algorithm. Thus, the RRT variant can be implemented in path planning domains that require a higher optimal path where time consumption is not a problem.

V .REFERENCES

[1] Ben Beklisi Kwame Ayawli, Xue Mei Mouquan Shen, Albert Yaw Appiah, Frimpong Kyeremeh “Optimised RRT-A* Path planning Method for Mobile Robots in Partially known Environment”

[2] Jiadong Li, Shirong Liu, Botao Zhang, Xiaodan Zhao “RRT-A* Motion Planning Algorithm for Non-Holonomic Mobile Robot”

[3] Christian Zammit and Erik-Jan van Kampen “Comparison between A* and RRT Algorithms for 3D UAV Path Planning”

[4] SUN-Qinpeng, Li-Meng, WANG- Tianhe, ZHAO-Chenpeng “UAV path planning based on Improved Rapidly-exploring Random Tree”