Path Planning in a Known Environment with Obstacles Using RRT*-SMART

Obaid ur Rahman
A. James Clark, School of Engineering
University of Maryland, College Park
College Park, MD 20742
Email: obdurhmn@umd.edu

Sagar Ojha
A. James Clark, School of Engineering
University of Maryland, College Park
College Park, MD 20742
Email: as03050@umd.edu

***Abstract*—Recently, several sampling-based algorithms have been introduced for motion planning. One such algorithm is Rapidly Exploring Random Tree (RRT), which is known for its speed and efficiency in finding obstacle-free paths. However, while RRT ensures probabilistic completeness, it does not guarantee finding the most optimal path. To address this limitation, Rapidly Exploring Random Tree Star (RRT*) was proposed as an extension of RRT. RRT* aims to achieve convergence towards the optimal solution, ensuring asymptotic optimality in addition to probabilistic completeness. However, it has been observed that RRT* can take an infinite amount of time to converge and has a slow convergence rate.**

In this report, we present the implementation of the RRT*-Smart algorithm, an extension of the RRT* algorithm, which is known for its ability to find obstacle-free paths efficiently. RRT*-Smart aims to overcome the limitations of RRT* by accelerating the convergence rate, enabling the algorithm to reach an optimal or near-optimal solution much faster. This is achieved through the incorporation of two fresh approaches to RRT*: path optimization and intelligent sampling. These methods improve the algorithm's effectiveness by choosing the best pathways and making wise choices during sampling. The simulation results of the suggested RRT*-Smart algorithm demonstrate its efficiency in a variety of obstacle-cluttered situations, as well as statistical and mathematical research. Overall, the implementation of RRT*-Smart offers a significant improvement over RRT* in terms of convergence rate and reduced execution time, thus providing a more practical solution for motion planning.

RRT*, RRT* - SMART, Intelligent Sampling, RRT

I. Introduction

The use of robots and automated systems has rapidly expanded in various industries, including manufacturing, healthcare, and agriculture. However, ensuring the safety and efficiency of robot movements in complex environments is a critical challenge that must be addressed. Motion planning is a fundamental problem in robotics that involves finding a feasible path for a robot to move from its initial position to a desired goal position, while avoiding collisions with obstacles.

Over the years, researchers have proposed various path planning algorithms to address this problem. Among these, sampling-based algorithms have become increasingly popular due to their computational efficiency and ability to find solutions without explicit information about obstacles.

Sampling-based algorithms rely on a collision checking module and construct a roadmap of feasible trajectories by connecting a set of points sampled from the obstacle-free space. Rapidly Exploring Random Tree (RRT) is one of the quickest algorithms to find an initial path [1], but it lacks asymptotic optimality. To overcome this limitation, RRT* was introduced as an extension of RRT, which optimizes the initial path and ensures asymptotic optimality [2]. However, RRT* still has limitations in terms of finite-time optimality and convergence rate.

To address these limitations, an extension of RRT* called RRT*-Smart was introduced. This algorithm incorporates intelligent exploration and sampling techniques to achieve faster convergence and reduced execution time. RRT*-Smart generates straighter paths with fewer waypoints, facilitating trajectory tracking and providing an efficient path planning solution. This report presents experimental results and performance analysis to demonstrate the distinguishing features of RRT*-Smart algorithm.

II. Brief Background of Motion Planning

In the past four decades, various algorithms have been proposed for motion planning, each with its own set of benefits and limitations. Geometric, grid-based, potential field, neural networks, genetic, and sampling-based methods are among them. Sampling-based algorithms, which do not require explicit knowledge of obstacles and are computationally efficient, have become more popular over time.

The Rapidly Exploring Random Tree (RRT) algorithm is one of the most efficient algorithms for finding an initial path, but it does not always provide the optimal path. To improve the efficiency of RRT, researchers have proposed several modifications such as potential function planner and density avoided sampling. However, the major breakthrough came with the development of RRT* in 2010 by Sertac Karaman and Emilio Frazzoli. RRT* is the first algorithm that is both probabilistically complete and asymptotically optimal. It starts by quickly determining an initial path and optimizes it as the number of samples increases. Compared to other algorithms like PRM* and RRT, RRT* has advantages in terms of both time and space complexity.

Despite its asymptotic optimality, RRT* is not able to achieve optimality in finite time and has a slow convergence rate [3]. To address these concerns, researchers have proposed an extension of RRT* called RRT*-Smart, which employs intelligent search space exploration techniques. RRT*-Smart leverages the initial path identified by RRT* and uses intelligent sampling to quickly obtain an optimal or near-optimal path with reduced execution time. As a result of having fewer waypoints, the path generated by RRT*-Smart is more efficient and allows for smoother robot trajectory tracking. The proposed algorithm is presented in this report through experiments and performance analysis.

III. Methodology

Rapidly-exploring Random Trees (RRT) and its extension RRT* are widely used algorithms in path planning for configuration spaces. These algorithms operate in a configuration space, denoted as Z, which represents the set of all possible transformations that can be applied to the robot. The configuration space encompasses the robot's state variables, such as position, orientation, and other relevant parameters. The problem at hand is to find a collision-free path within the configuration space that connects the initial state, denoted as z_{init}, to the goal state, denoted as z_{goal}, while adhering to various constraints and objectives.

In this context, the RRT* (Rapidly-exploring Random Tree-Star) algorithm and its extension RRT*-Smart have emerged as powerful techniques for solving path planning problems, denoted as $Z \subset \mathbb{R}^n$, where n represents the dimensionality of the search space. These algorithms aim to efficiently explore the configuration space and find both feasible and optimal paths. RRT builds a rapidly exploring tree structure by randomly sampling points in the configuration space and iteratively expanding the tree towards unexplored regions. It employs cost heuristics such as cost-to-come or cost-to-go to guide the search towards promising areas, resulting in feasible paths.

RRT*-Smart, on the other hand, is an enhancement of the RRT* algorithm that incorporates path optimization process along with intelligent sampling. It identifies redundant points in the tree and removes them to simplify the path while preserving the optimality. Additionally, RRT*-Smart introduces the concept of beacon nodes, which are points along the optimized path. These beacon nodes serve as landmarks for intelligent sampling, where the algorithm biases the exploration within the vicinity of these key points. The biasing is controlled by parameters known as the biasing radius and the biasing ratio. The region occupied by obstacles is represented by $Z_{\text{obs}} \subset Z$, while the obstacle-free region is denoted as $Z_{\text{free}} = Z \setminus Z_{\text{obs}}$. The starting point is denoted by $z_{\text{init}} \in Z_{\text{free}}$, and the goal point is represented as $z_{\text{goal}} \in Z_{\text{free}}$. By focusing the search on regions near the beacon points, RRT*-Smart effectively improves the exploration efficiency and optimizes the generated paths.

The combination of intelligent sampling, path optimization, and the inherent exploration capabilities of RRT* make RRT*-Smart a valuable tool for path planning in complex environments. It provides a balance between feasibility and optimality, allowing robots to navigate safely and efficiently in real-world scenarios. By understanding the principles and methodologies behind RRT*, RRT*-Smart, and related algorithms, researchers and engineers can develop advanced path planning solutions to address the challenges of autonomous navigation and robotic systems.

A. RRT*-Smart Objectives

The algorithm achieves the final objective, path optimization with efficient exploration, by achieving 3 smaller objectives: feasible path, optimal path, and convergence to optimal path.

1) Feasible Path Solution: The first objective of the algorithm is to find a feasible path from the initial state z_{init} to the goal state z_{goal} in the obstacle-free region Z_{free}. A feasible path σ is a continuous function mapping the interval $[0, s]$ to Z_{free} such that $\sigma(0) = z_{\text{init}}$ and $\sigma(s) \in z_{\text{goal}}$. In other words, the path should start from the initial state and reach a state within the goal region. If no such feasible path exists, the algorithm should report failure.

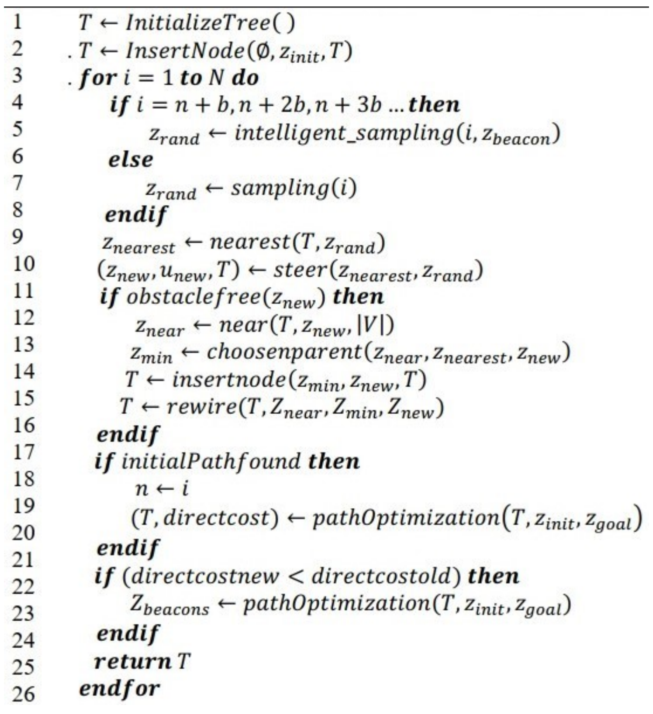
2) Optimal Path Solution: The second objective is to find an optimal path between the initial state z_{init} and the goal state z_{goal} within the obstacle-free region Z_{free}. An optimal path σ^* is a function that maps the interval $[0, s]$ to Z_{free} that connects the initial state to the goal state while minimizing the overall path cost. The cost $C(\sigma)$ of a path σ is calculated as the sum of costs $C(\sigma_i)$ associated with each segment or edge in the path. The objective is to find the path σ^* that minimizes the total path cost, where $C(\sigma) = \sum C(\sigma_i)$.

3) Convergence to Optimal Solution: The third objective is to achieve convergence to the optimal solution in the shortest possible time. The algorithm aims to find the optimal path σ^* in the obstacle-free region Z_{free} within a given time limit t. The time limit represents the maximum allowable computation time for the algorithm to converge to the optimal solution. The algorithm should utilize efficient search and optimization techniques to find the optimal path within the specified time constraint.

By addressing these three problems, RRT*-Smart, as well as other path planning algorithms [4] can effectively navigate the configuration space Z and find collision-free paths from the initial state to the goal state. The goal is to achieve both feasibility and optimality in path planning, ensuring that the generated paths are obstacle-free, efficient, and cost-effective.

B. Algorithm Procedure

The algorithm for RRT*-Smart is shown in Fig. 1. The algorithm follows a series of steps to achieve efficient and optimal path. Step 1 shows initialization of a tree data structure, T, to store the states of the robot. Note that "state" is also referred to as "node". T is initialized with the initial state of the robot, z_{init}. The algorithm iterates for a set maximum number of iteration, sampling random points, z_{random}, in the configuration space (steps 5 & 7). The random sampling takes place to get random node in the free space until the initial path is found. Once the initial path is found, both random and biased samplings take place. The biased sampling takes place as specified by a factor called "biased ratio", b. Step 4 shows that the biased sampling repeats at the iteration that is the multiple of the biasing ratio b. The sampling is biased within "biasing radius", b_{radius}, of the "beacon points", z_{beacon}. z_{beacons} are regions where the straight path to z_{goal} are broken. Biasing the sampling around z_{beacon} further optimizes the path.



```
1    $T \leftarrow InitializeTree(\,)$
2   . $T \leftarrow InsertNode(\emptyset, z_{init}, T)$
3   . for $i = 1$ to $N$ do
4        if $i = n + b, n + 2b, n + 3b \ldots$ then
5            $z_{rand} \leftarrow intelligent\_sampling(i, z_{beacon})$
6        else
7            $z_{rand} \leftarrow sampling(i)$
8        endif
9        $z_{nearest} \leftarrow nearest(T, z_{rand})$
10       $(z_{new}, u_{new}, T) \leftarrow steer(z_{nearest}, z_{rand})$
11       if $obstaclefree(z_{new})$ then
12           $z_{near} \leftarrow near(T, z_{new}, |V|)$
13           $z_{min} \leftarrow choosenparent(z_{near}, z_{nearest}, z_{new})$
14           $T \leftarrow insertnode(z_{min}, z_{new}, T)$
15           $T \leftarrow rewire(T, Z_{near}, Z_{min}, Z_{new})$
16       endif
17       if $initialPathfound$ then
18           $n \leftarrow i$
19           $(T, directcost) \leftarrow pathOptimization(T, z_{init}, z_{goal})$
20       endif
21
22       if $(directcostnew < directcostold)$ then
23           $Z_{beacons} \leftarrow pathOptimization(T, z_{init}, z_{goal})$
24       endif
25       return $T$
26   endfor
```

Fig. 1. The algorithm for RRT*-Smart. Image credit: [4]

The nearest node, z_{nearest}, from z_{random} is found in step 9. An action of a fixed step-size (usually the step size is 1) that leads the robot towards z_{random} from z_{nearest} is performed. The resulting node, z_{new}, and its corresponding action, u_{new}, are stored and T is updated accordingly (step 10). T is updated only with the information of z_{new}. The parent of z_{new} is updated in the subsequent steps. Fig.2 illustrates the process in step 10.

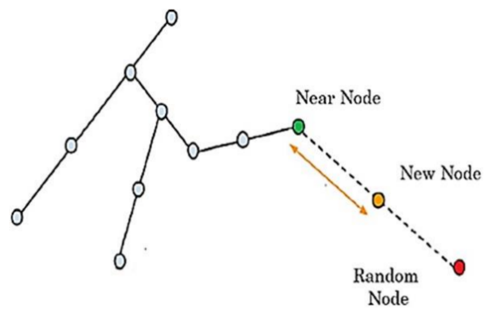


Fig. 2. New node being created in the tree from the nearest node (Near Node) to random node. Image credit: [5]

If z_{new} is in the obstacle-free region, the nearby nodes, z_{near}, within the specified step-size of z_{new} are identified and stored (step 12). Fig.3 illustrates the process in step 12.

The parent node, z_{min}, that results to the least cost-to-come for z_{new} is selected from z_{near}. Subsequently, z_{new} with its

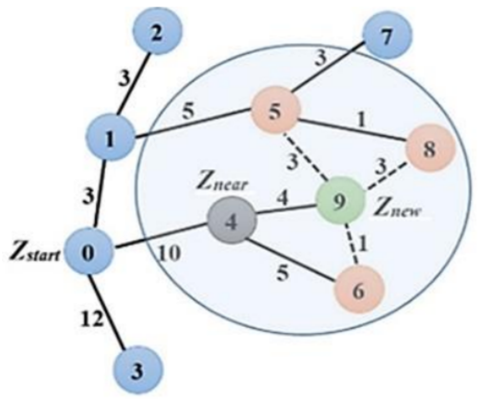


Fig. 3. Nodes that are within the specified radius from new node, Z_{new} or Node 9, are identified (Nodes 4, 5, 6, 8). Image credit: [5]

parent, z_{min}, is stored in T (step 14). Fig.4 illustrates the process in step 14.

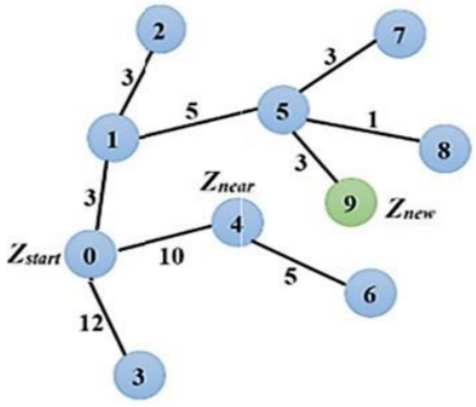


Fig. 4. Node 5 results to the lowest cost-to-come for Z_{new}. Hence, the parent node to Z_{new} is Node 5. Image credit: [5]

The tree, T, is updated to improve the existing paths in step 15. This is also called rewiring the tree. If the existing cost-to-come for a node, z_i, in z_{near} is greater than the cost-to-come to z_i if the path is through z_{new}, then z_{new} is updated as the parent node for z_i. Note that z_i is not shown in Fig.1. The rewiring process results in the optimal path to z_i. In fact, the paths from z_{init} to all the nodes are optimized. Fig.5 illustrates the process in step 15.

If an initial path from z_{init} to z_{goal} is found, then path optimization is executed to remove redundant nodes and improve optimality (step 19). Fig. 6 shows the process of removing redundant nodes.

The farthest visible nodes are considered to get the direct cost to reach z_{goal}. This complies with the rules of the sides of a triangle. Hence, the path obtained using farthest visible nodes is always optimal compared to other path that traverses through the same nodes. Fig. 7 shows that the path in green is shorter than the path in black. Hence, the cost is also smaller for green path.

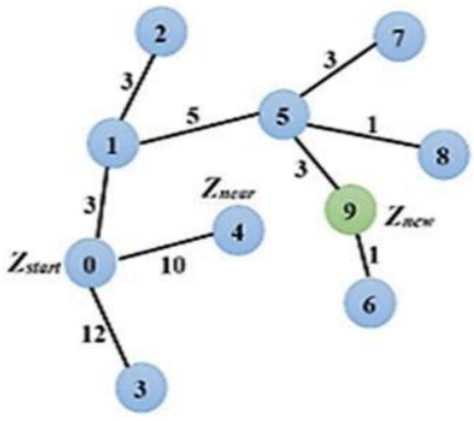


Fig. 5. The parent of Node 6 is updated to Z_{new} from Node 4. The previous cost-to-come for Node 6 was 15 and the new cost-to-come is 9. Image credit: [5]

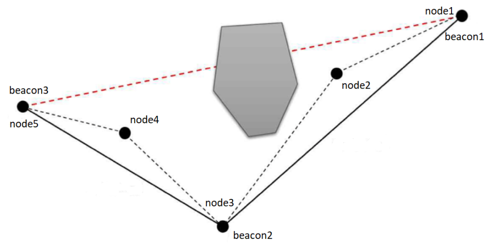


Fig. 6. The beacons are the nodes which are farthest and are visible. They are also the places where straight path is broken and they form the region for intelligent sampling.

The cost of the path from step 19 is computed, and if it is lower than the cost of the previous path, then the "beacon nodes", Z_{beacons}, are updated. Z_{beacons} would thus contain the new beacons that would result in shorter path than previous iteration. Finally, the algorithm returns the final tree that contains the optimal path from z_{init} to z_{goal}.

C. Algorithm Implementation

The algorithm implemented in this paper is similar to the algorithm in Fig. 1. However, the implmentation of the data-structure is primarily different. A 3-dimensional Numpy array is used as the data structure which contrasts with the tree data structure as mentioned in the algorithm. Hence, the configuration space is discretized. This results in a strightforward implementation while extracting and updating the information of the nodes such as in steps 9, 12, 14, and 15. Moreover, the obstacle avoidance implementation is also easier.

The first two dimensions of the array are the node data, i.e. the position of the robot, and the third dimension stores the information about the parent node and cost-to-come for the particular node. The algorithm is implemented on a point robot with the step-size of 1. The environment used is similar to the one used by the original authors and several others.

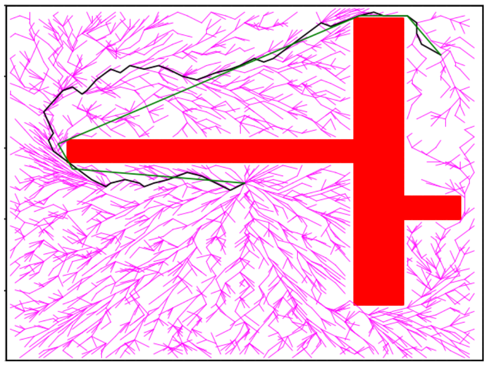


Fig. 7. Path shown in green is shorter than the path shown in black. Green path is formed by connecting the farthest visible node. Green path is obtained from RRT*-Smart and black path from RRT* with intelligent sampling. Pink lines connect parent and child nodes that are explored using RRT*-Smart.

IV. Results

The final paths obtained from the RRT*-Smart are plotted using the "matplotlib" library in Python as shown in figures 7, 8, 9, 10, 11. The key parameters within the algorithm are changed to observe the effects on the solution path obtained by the algorithm. Table I shows the final cost of the path when the parameters are changed. The parameters include maximum iteration number i, search radius $|V|$ to get the nearby nodes (z_{near}) of z_{new}, robot step-size u_{size}, biasing ratio b, and biasing radius b_{radius}. Experiment 1 is chosen to be the standard experiment and other experiment data and plots are compared against experiment 1 data and plots respectively.

Experiment	i	$\|V\|$	u_{size}	b	b_{radius}	$i_{\text{initial\_path}}$	$C_{\text{RRT*}}$	$C_{\text{RRT*-Smart}}$
1	3000	3	3	5	5	1950	206.4	139.4
2	5000	3	3	5	5	1950	206.4	126.5
3	3000	5	5	5	5	1601	178.1	140.5
4	3000	3	3	2	5	1950	206.4	139.1
5	3000	3	3	5	2	1950	206.4	138.7

TABLE I
THE ITERATION AT WHICH INITIAL PATH IS FOUND $i_{\text{INITIAL\_PATH}}$, THE COSTS OF THE INITIAL PATH OBTAINED FROM RRT* $C_{\text{RRT*}}$ AND RRT*-SMART FOR THE FINAL PATH $C_{\text{RRT*-SMART}}$ FOR THE GIVEN PARAMETERS: MAXIMUM ITERATION NUMBER i, SEARCH RADIUS ($|V|$), ROBOT STEP-SIZE (u_{SIZE}), BIASING RATIO b, AND BIASING RADIUS b_{RADIUS}. THE START AND GOAL NODES ARE (50, 50) AND (95, 85). THE ALLOWABLE MAXIMUM ERROR FROM THE SET GOAL IS 2 UNITS.

Figure 8 shows a significantly different optimal path than figures 7, 9, 10, and 11. The iteration number is increased compared to others. As a result, more of the free space gets explored. Hence, the probability of finding better solution increases. Thus, experiment 2 has the lowest total cost of the path obtained from RRT*-Smart.

Figure 9 shows the effect of increased step-size and subsequently increased search radius for the nearby nodes. Consequently, the rewiring effected a larger region and so the path is smoother compared to other experiments. This resulted in the lowest cost of the path when the initial path was obtained. Also, due to a larger step-size, more area is explored.

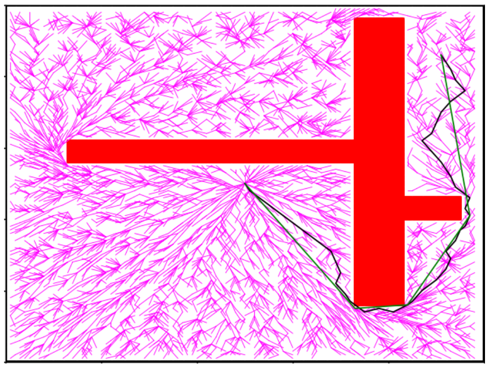


Fig. 8. Plot for experiment 2 in Table I. Green path is obtained from RRT*-Smart and black path from RRT* with intelligent sampling.

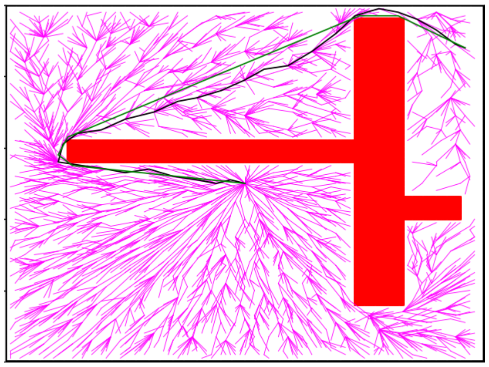


Fig. 9. Plot for experiment 3 in Table I. Green path is obtained from RRT*-Smart and black path from RRT* with intelligent sampling.

Figure 10 shows the effect of decrease biasing ratio on the optimal path. Biasing ratio dictates the frequency of intelligent sampling once the initial solution is obtained. Since the biasing ratio is small, the less of the free space is explored and more of the regions nearby the beacon points are explored. The cost of the final path is not significantly different that for experiment 1. In this scenario, RRT* would perform better than the RRT* with intelligent sampling (constant biasing ratio) given a high maximum iteration value. Also, dynamic biasing could be useful in this scenario for better performance [3].

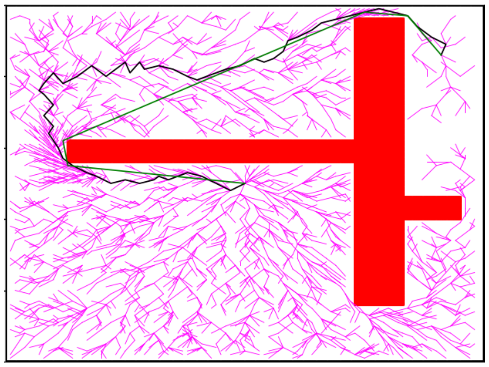


Fig. 10. Plot for experiment 4 in Table I. Green path is obtained from RRT*-Smart and black path from RRT* with intelligent sampling.

Figure 11 depicts experiment 5 where the biasing radius is decreased. This resulted in a greater exploration of regions around the sharp angles of the obstacle. Therefore, the local shortest paths around the corners of the obstacles are parallel to the edges. The cost is therefore the lower than the other paths going from the left side of the obstacle.

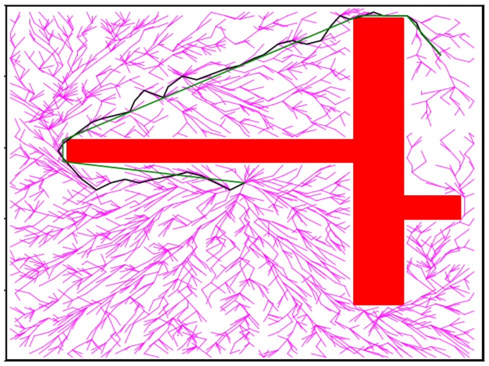


Fig. 11. Plot for experiment 5 in Table I. Green path is obtained from RRT*-Smart and black path from RRT* with intelligent sampling.

The optimal path obtained from the algorithm is also dependent upon the random sampling. Hence, the cost of the final path obtained by altercation of parameters might not be significantly different. However, the results obtained and analysis described above are valid for a large number of experiments. Similar analysis can be performed to adjust the parameters to get the better solution.

V. Conclusion

By combining the techniques of random sampling (RRT) and rewiring (RRT*), along with the added benefits of intelligent sampling and connecting the farthest visible nodes, RRT*-Smart emerges as a powerful algorithm that greatly improves the efficiency and optimality of path planning in complex environments. The integration of random sampling allows the algorithm to explore a wide range of configuration space, while the rewiring step optimizes the connections and paths within the constructed tree. Additionally, the intelligent sampling approach, which includes biasing the sampling towards specific regions near beacon nodes, enhances the algorithm's ability to obtain better optimal solution around the obstacles.

The efficiency and optimality offered by RRT*-Smart make it a highly valuable tool in various domains. In the field of robotics, the algorithm can assist in determining optimal paths for robots to navigate in challenging environments, ultimately improving their overall performance and safety. In autonomous navigation systems, RRT*-Smart can play a crucial role in helping vehicles plan their trajectories efficiently, ensuring smooth and reliable transportation. Furthermore, in motion planning applications, such as virtual simulations and virtual reality environments, RRT*-Smart can guide the movement of virtual characters or objects, providing realistic and optimized paths.

The future works include the implementation of the RRT*-Smart algorithm on robots with different actions and constraints. The visualization would be carried out on simulation software frameworks such as Gazebo and Webots. Also, the algorithm would be implemented on physical robot.

References

[1] LaVelle S, et al. "Rapidly-Exploring Random Trees : A New Tool for Path Planning."

[2] Karaman, S, and Emilio F. "Sampling-Based Algorithms for Optimal Motion Planning."

[3] Nasir J, Islam F, Malik U, et al. "RRT*-SMART: A Rapid Convergence Implementation of RRT*". International Journal of Advanced Robotic Systems. 2013;10(7). doi:10.5772/56718

[4] Suwoyo, Heru, et al. "An Integrated Rrt*smart-A* Algorithm for Solving the Global Path Planning Problem in a Static Environment." IIUM Engineering Journal.

[5] Noreen I, et al. "A Comparison of RRT, RRT* and RRT* -Smart Path Planning Algorithms". IJCSNS

[6] Tak, H.T. "Improvement of RRT*-Smart Algorithm for Optimal Path Planning and Application of the Algorithm in 2 & 3-Dimension Environment". Journal of Korean Society for Aviation and Aeronautics.