A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms

Iram Noreen[1], Amna Khan[2], Zulfiqar Habib[3]

Department of Computer Science, COMSATS Institute of Information Technology, Lahore.

Summary
Sampling based planning algorithm such as RRT and RRT* are extensively used in recent years for path planning of mobile robots. They are probabilistic complete algorithms and have natural support for solving high dimensional complex problems. RRT*-Smart is an extension of RRT* with faster convergence as compared to its predecessors. This paper provides an analytical review of the three algorithms. Impact of different parameters on algorithm's performance is also evaluated. Moreover, a performance comparison for different optimality criteria such as path cost, run time and total number of nodes in tree is performed through simulation based experiments. Further, the comparative analysis is concluded with future research directions.

Key words:
Path Planning, RRT, RRT, mobile robots, Comparison, Review.*

1. Introduction

Path Planning for mobile robots has many useful applications in autonomous cars [1], Unmanned Aerial Vehicles (UAVs) [2], industrial forklifts [3], surveillance operations and planetary space missions [4, 5]. Path planning algorithms aim to find a collision free path from an initial state to a goal state with optimal or near optimal path cost. Sampling Based Planning (SBP) algorithms have been extensively used for path planning of mobile robots in recent years [5, 6] . SBP algorithms are known to provide quick solutions for complex and high dimensional problems using randomized sampling in search space [6, 7]. However SBPs are probabilistic complete, i.e., a solution will be provided, if one exists, given infinite runtime [6, 7]. Lavelle [8] proposed Rapidly exploring Random Tree (RRT) [8, 9], which is a well-known SBP algorithm. RRT supports dynamic environment and non-holonomic constraints for car like robots [9] very well. Lavelle applied it successfully to problems comprising of up to twelve degrees of freedom with both holonomic and non-holonomic constraints. However, path generated by RRT was not optimal. Karaman and Frazzoli [5] proposed a variation of RRT called RRT* with proven asymptotically optimal property. RRT* improved path quality by introducing major features of tree rewiring and best neighbor search. However, it obtained asymptotic optimality at the cost of execution time and slower path convergence rate. Nasir et al. presented RRT* Smart [10] with main focus on improving convergence rate of RRT*. Two major features introduced by RRT*-Smart called intelligent sampling and path optimization improved path cost and convergence rate.

RRT and RRT* have numerous successful applications in robotics. Significant body of research has addressed the problem of optimal path planning using RRT* in recent years. These methodologies have gained tremendous success in solving single-query high dimensional complex problems. This paper presents a simulation based experimental comparison of the aforementioned algorithms in an environment cluttered with obstacles. Effect of different parameters on the performance of these approaches is discussed. Further, limitations and future directions for improvement are also suggested.

Next section describes the methodology of RRT, RRT* and RRT*-Smart path planning approaches. Experimental results along with comparative analysis are presented in Section 3. Section 4 discusses future recommendations, followed by conclusion in the last section.

2. Algorithms Overview

2.1 Problem Definition

RRT, RRT*, and RRT*-Smart operate in a configuration space, which is set of all possible transformations that could be applied to the robot [4, 11]. Let the given state space be denoted by a set $Z \subset \mathbf{R}^n, n \in \mathbf{N}$ where n represents the dimension of the given search space. The area of the search space which is occupied by obstacles is represented by $Z_{obs} \subset Z$ and region free from obstacles is represented by $Z_{free} = Z / Z_{obs}$. $z_{goal} \subset Z_{free}$ represents goal and $z_{init} \subset Z_{free}$ represents starting point. z_{init} and z_{goal} are provided to planner as inputs. The problem is to find a collision free path between initial z_{init} and goal

Manuscript received October 5, 2016
Manuscript revised October 20, 2016

z_{goal} states in Z_{free}, in least possible time $t \in \mathbf{R}$, with minimum path cost.

Problem 1 (Feasible Path Solution): Find a path $\sigma_f : [0, s]$, if one exists in $Z_{free} \subset Z$ such that $\sigma_f(0) = z_{init} \subset Z_{free}$ and $\sigma_f(s) \in z_{goal}$ and report failure if no such path exists.

Problem 2 (Optimal Path Solution): Find an optimal path $\sigma_{*f} : [0, s]$ which connects z_{init} and z_{goal} in v, such $Z_{free} \subset Z$ such that the cost of the path σ_{*f} is minimum, $C(\sigma_{*f}) = \{\min_{\sigma s} C(\sigma_f) : \sigma_f \in \sum f\}$.

Problem 3 (Convergence to Optimal Solution): Find an optimal path $\sigma_{*f} : [0, s]$ in $Z_{free} \subset Z$ in the least possible time $t \in \mathrm{R}$.

2.2 RRT

RRT constructs a tree using random sampling in search space. The tree start from an initial state say z_{init} and expands to find a path towards goal state say z_{goal}. The tree gradually expands as the iteration continue. During each iteration, a random state say z_{rand} is selected from configuration space Z. If random sample z_{rand} lies in obstacle free region, then a nearest node say $z_{nearest}$ is searched in tree according to a defined metric ρ. If z_{rand} is accessible to $z_{nearest}$ according to predefined step size, then tree is expanded by connecting z_{rand} and $z_{nearest}$. Otherwise, it returns a new node z_{new} by using a steering function, thus expands tree by connecting z_{new} with $z_{nearest}$. A Boolean collision checking process is performed to ensure collision free connection between tree nodes z_{new} and $z_{nearest}$. When an initial path is found even then the process continues until a predefined time period expires or fixed number of iterations are executed. Node expansion process is described in Figure 1. RRT algorithm is presented in Table 1.

Table 1: RRT Algorithm.

Algorithm 1.
T = (V, E) ← RRT(z_{init})

1 T ← InitializeTree();
2 T ← InsertNode(Ø, z_{init}, T);
3 ***for*** i=0 to i=N ***do***
4 z_{rand} ← Sample(i);
5 $z_{nearest}$ ← Nearest(T, z_{rand});
6 (z_{new}, U_{new}) ← Steer ($z_{nearest}$, z_{rand});
7 ***if*** Obstaclefree(z_{new}) ***then***
8 T ← InsertNode(z_{min}, z_{new}, T);
9 ***return*** T

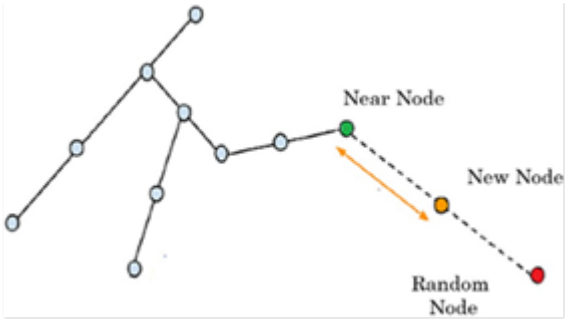


Figure 1: Tree expansion process.

Further detail of some major functions is described as the following:-

Sample: This function generates a random positon z_{rand} from search space in obstacle free region Z_{free}.

Nearest: This function returns the nearest node from $T = (V, E)$ to z_{rand} according to a cost function.

Steer: This function provides a control input u [0, T] which drives the system from $z(0) = z_{rand}$ to $z(T) = z_{nearest}$ along the path z:[0,T] → Z giving z_{new} at a distance Δq from $z_{nearest}$ towards z_{rand} where Δq is the incremental distance.

CollisionCheck: This function is used to check collision detection of a tree branch and returns true if it lies in obstacle free region, i.e., whether a path z:[0, T] lies in the Z_{free} for all t=0 to t=T.

Near: This function returns the nearby nodes in tree defined by (1).

InsertNode: This function adds a node z_{new} to V in the tree $T = (V, E)$ to connect node z_{min} as its parent.

2.3 RRT*

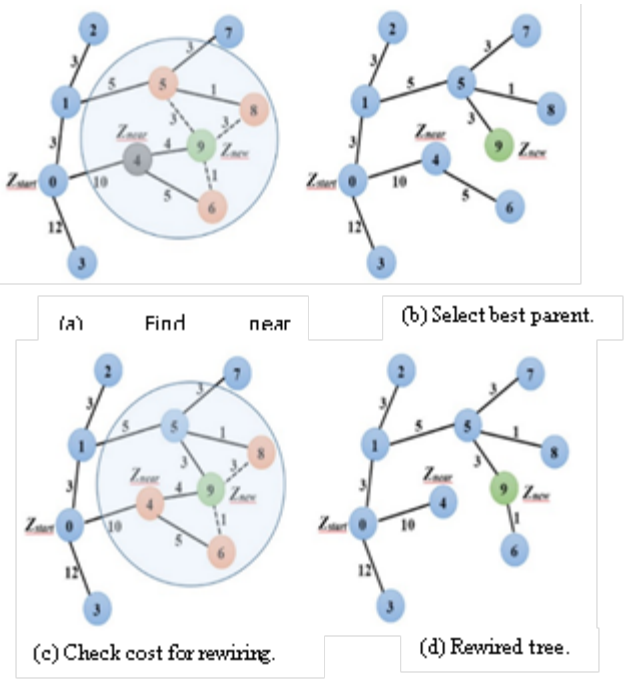


Figure 2: Near neighbour search and rewiring operations in RRT*.

RRT* inherits all the properties of RRT and works similar to RRT. However, it introduced two promising features called near neighbor search and rewiring tree operations [5]. Near neighbor operations finds the best parent node for the new node before its insertion in tree. This process is performed within the area of a ball of radius defined by

$$k = \gamma\left(\frac{\log(n)}{n}\right)^{\frac{1}{d}}, \qquad (1)$$

where d is the search space dimension and γ is the planning constant based on environment. Rewiring operation rebuilds the tree within this radius of area k to maintain the tree with minimal cost between tree connections as shown in Figure 2 [12]. Space exploration and improvement of path quality is shown in Figure 3. As the number of iterations increase, RRT* improves its path cost gradually due to its asymptotic quality, whereas RRT does not improves its jaggy and suboptimal path.

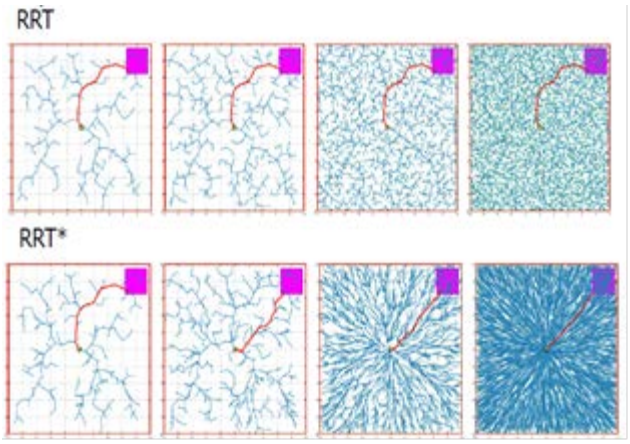


Figure 3: Dense space exploration and path refinement in RRT* in comparison to RRT [5].

Due to increased efficiency to get less jagged and shorter path, features of rewiring and neighbor search are being adapted in recent revisions of RRT*. However, these operations have an efficiency trade-off. Though, it improved path cost but on the other hand it also slowed down convergence rate of RRT*. The details of the two new features introduced in RRT* are as follows:-

Rewire: This function checks if the cost to the nodes in z_{near} is less through z_{new} as compared to their older costs, then its parent is changed to z_{new}.

ChooseParent: This function selects the best parent z_{new} from the nearby nodes.

RRT* algorithm is described in Table 2.

Table 2: RRT Algorithms

Algorithm 2.
T = (V, E) ← RRT*(z_{ini})

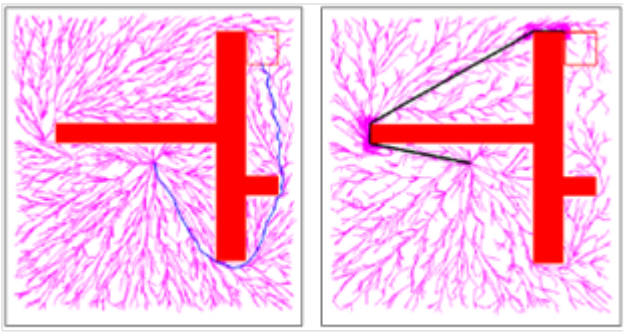
1 T ← InitializeTree();
2 T ← InsertNode(Ø, z_{init}, T);
3 ***for*** i=0 to i=N ***do***
4 z_{rand} ← Sample(i);
5 $z_{nearest}$ ← Nearest(T, z_{rand});
6 (z_{new}, U_{new}) ← Steer ($z_{nearest}$, z_{rand});
7 ***if*** Obstaclefree(z_{new}) ***then***
8 z_{near} ← Near(T, z_{new}, |V|);
9 z_{min} ← Chooseparent (z_{near}, $z_{nearest}$, z_{new});
10 T ← InsertNode(z_{min}, z_{new}, T);
11 T ← Rewire (T, z_{near}, z_{min}, z_{new});
12 ***return*** T

2.4 RRT*-Smart

RRT*-Smart is an extended version of RRT* and executes similar to RRT*, however it performs a path optimization process when an initial path is found. This optimization process removes redundant nodes from the initial path found. Moreover, it also identifies beacon nodes for path improvement. Second major feature introduced by RRT*-Smart is intelligent sampling. This sampling is different from random sampling because it is biased towards beacon nodes of optimized path. It uses a Biasing Radius to set radius for intelligent exploration around selected beacons. As soon as RRT*-Smart finds a shorter path, it performs path optimization process again to generate new beacon nodes. Thus, RRT*-Smart accelerates the path convergence and improves path cost. Further, it uses a Biasing Ratio to intelligently select either of Uniform Sampling or Intelligent Sampling. RRT*-Smart exhibits better path convergence than RRT*, as shown in Figure 4. However, its Biasing Ratio is either manually set by programmer or can be automated using

$$BiasingRatio = \left(\frac{n}{z_{free}} \right) * B, \qquad (2)$$

where B is again a programmer dependent constant. However, Biasing Ratio has a trade-off between rate of convergence and rate of exploration of search space.



(a) RRT* cost = 574. (b) RRT*-Smart cost = 540.

Figure 4: Convergence at iteration n = 4200 [10].

RRT*-Smart algorithm is described in Table 3.

3. Experiments and Simulation Results

In this section, analysis of three algorithms RRT, RRT*, RRT* Smart is presented. To evaluate their performance a simulation environment is developed using 64-bit MATLAB version 15. The operating system used is 64-bit Windows 8.1 Pro. Test cases of simulation are executed on a PC with an Intel i7-4790 CPU @ 3.60 GHz and 8GB internal RAM. Three different types of experiments are performed as described Plater in this section. Purpose of simulation experiments in this section is as follows:-

Table 3: RRT*-Smart Algorithm.

Algorithm 3.
T = (V,E) ← RRT*-Smart(z_{ini})

1 $T \leftarrow$ InitializeTree();
2 $T \leftarrow$ InsertNode(Ø, z_{init}, T);
3 ***for*** i=0 to i=N ***do***
4 ***if*** i=n+b, n+2b, n+3b…. ***then***
5 $z_{rand} \leftarrow$ Sample(i, $z_{beacons}$);
6 ***else***
7 $z_{rand} \leftarrow$ Sample(i);
8 $z_{nearest} \leftarrow$ Nearest(T, z_{rand});
9 (z_{new}, U_{new}, T_{new}) ← Steer ($z_{nearest}$, z_{rand});
10 ***if*** Obstaclefree(z_{new}) ***then***
11 $z_{near} \leftarrow$ Near(T, z_{new}, $|V|$);
12 $z_{min} \leftarrow$ Chooseparent (z_{near}, $z_{nearest}$, z_{new});
13 $T \leftarrow$ InsertNode(z_{min}, z_{new}, T);
14 $T \leftarrow$ Rewire (T, Z_{near}, Z_{min}, Z_{new});
15 ***if*** InitialPathFound ***then***
16 $n \leftarrow$ i;
17 (T, *directcost*) ← PathOptimization(T, z_{init}, z_{goal});
18 ***if*** (*directcostnew* < *directcostold*) ***then***
19 $Z_{beacons} \leftarrow$ PathOptimization(T, Z_{init}, Z_{goal});
20 ***return*** T

- To highlight the RRT* sensitivity with rewiring radius obtained using different values of γ, as defined by (1).
- To highlight the sensitivity of smart heuristics of Biasing Ration and Biasing Radius in RRT*-Smart, defined by (2).
- To highlight performance difference between RRT, RRT* and RRT*-Smart with respect to different performance parameters e.g. path cost, runt time, and as well as tree density.

In these experiments the path cost is calculated in terms of Euclidean distance defined by

$$\Delta d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (3)$$

3.1. Effect of γ on RRT* Performance

Experiments were carried out for 10 different trials with 3000 iterations and different γ values. Simulation results showed that increased value of γ increases rewiring radius which in turn effects the convergence and path cost as well. Figure 5 shows path cost and average execution time analysis with γ values of 20, 10 and 5. It is evident that with increase of γ value, path cost and execution time are decreased.

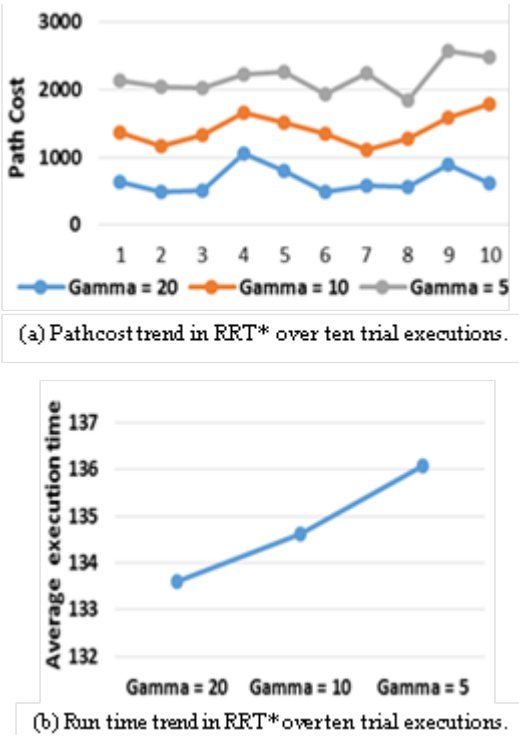


Figure 5: Comparison of different γ values on performance of RRT*.

3.2. Effect of Biasing Radius on RRT*-Smart Performance

Search space exploration in RRT*-Smart is heavily effected by its Biasing Radius heuristic. An increased value of Biasing Radius increases the exploration around beacon as shown in Figure 6. A suitable range of Biasing Radius is adjusted to get the minimum path cost with maximum exploration of best nodes which varies for different environment maps.

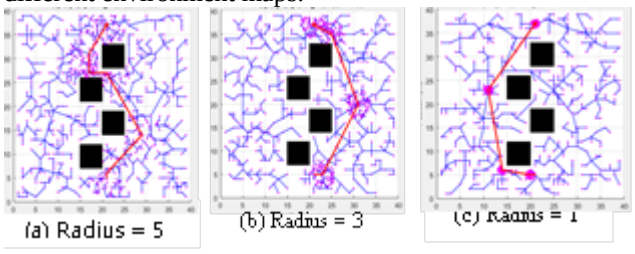


Figure 6: Effect of Biasing Radius in RRT*-Smart.

Convergence rate and path cost of RRT*-Smart is also very sensitive to Biasing Ratio. Figure 7 shows the average number of nodes and average path cost for 10 independent runs with fixed 2000 iterations. This experiment was repeated three time using different values of constant B in Biasing Ratio from (2), as shown in Figure 7. This constant B requires fine tuning to identify the suitable range of value according to the application needs. Suitable value is the one which could balance the trade-off between uniform exploration and intelligent exploration in search space.

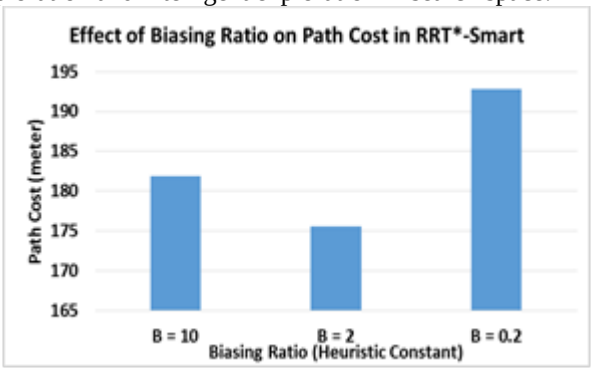


Figure 7: Impact of Biasing Ratio in RRT*-Smart.

3.3. Performance Comparison

Figure 8 shows results of one sample run executed with 8000 iterations for three different environment maps. Results show that RRT* and RRT*-Smart improve initial path as compared to RRT remarkably. However, they both take more execution time than basic RRT. This is due to the fact that RRT* and RRT*-Smart use two additional operations than RRT i.e., rewiring tree and best neighbor search. These two features improve the path cost to generate less jaggy and shorter path. On the other hand, these features also slow down the convergence rate and increase computational time. Further, RRT*-Smart evidently takes less time than RRT*. Its optimization operation and intelligent sampling operation converge it quickly to shorter path in less number of iterations than RRT*.

Analysis of results in Figure 8 also showed that RRT*-Smart converges quickly than both other approaches. However, it also suffers from large number of nodes like RRT and RRT*. Because switching to intelligent sampling in RRT*-Smart also generates denser tree in identified beacons area, as shown in Figure 8. In all these approaches, tree gets populated with such nodes which do not contribute in the final solution. They merely increase the tree density and memory requirements consequently effecting computational time. Developing operations to limit tree nodes to maximum useful nodes could increase the efficiency of these approaches.

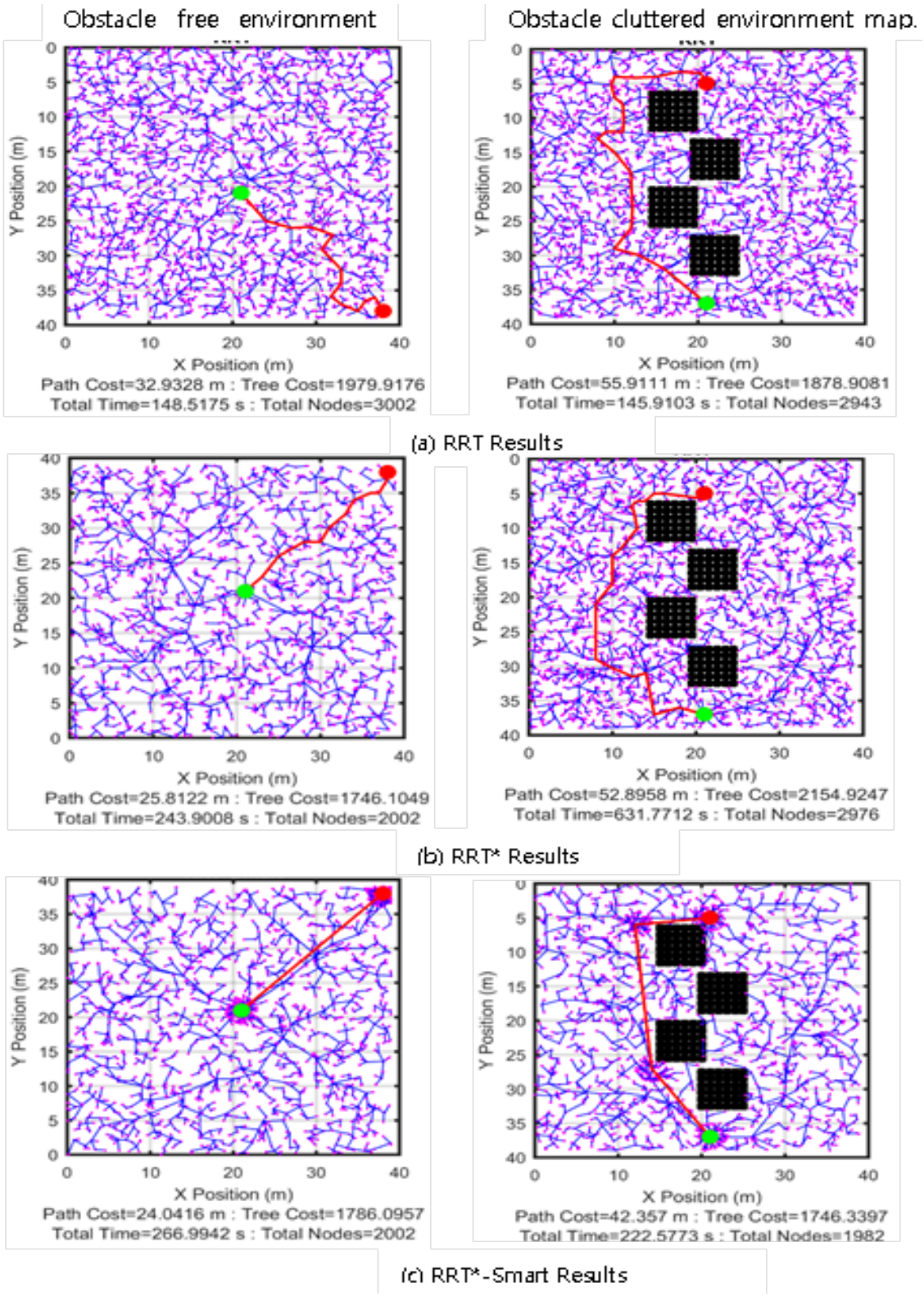


Figure 8: Path quality comparison of three algorithms with 8000 iterations in a sample run.

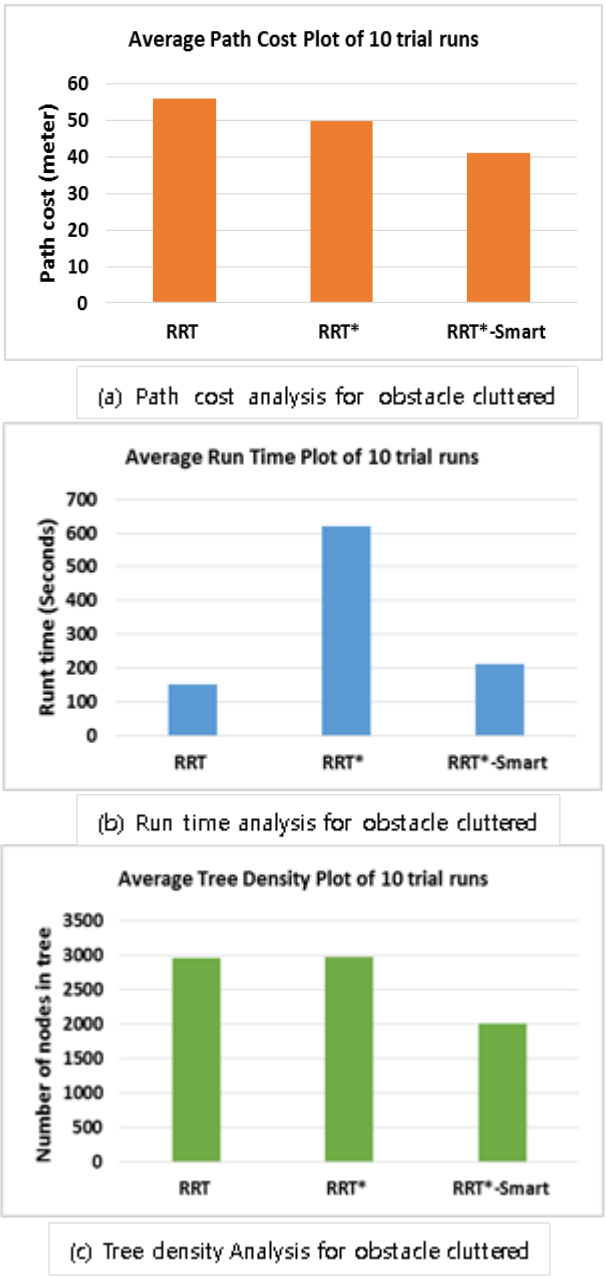


Figure 9: Average Run Time and Path Cost Comparison with 8000 iterations in 10 trial runs.

Path cost, run time and tree density plots after numerical analysis of executing all three algorithms on obstacle cluttered map are shown in Figure 9. The plots demonstrate that RRT*-Smart generates shorter path with less path cost than RRT and RRT* for 10 different runs of 8000 iterations. Second fact that is evident from the evaluation of execution time trend is that RRT has significantly less time than both other algorithms. Because RRT* and RRT*-Smart offer near neighbor search and rewiring operations, which increase execution time per iterations than basic RRT. However, they generate significantly shorter paths with much less path cost than basic RRT as shown in Figure 8 and in Figure 9. Third, the exploration of search space and nodes expansion in tree is also linked with memory requirements and computational resources. RRT*-Smart expands tree rapidly as compared to RRT and RRT* because of its intelligent sampling and path optimization processes. Hence it generates shorter path in reasonable time using less dense tree thus saving the computational resources. Table 4 presents the comparative summary of three approaches for different performance parameters.

Table 4: Summary of Comparative Analysis

Parameter	***RRT***	***RRT****	***RRT*-Smart***
Path Cost (m)	56	50	41
Run Time (Sec)	150	621	210
Tree Density (number of nodes in tree)	2954	2972	2000
Completeness	Probabilistic	Probabilistic	Probabilistic
Optimality	Non-optimal	Asymptotic Optimal	Asymptotic Optimal

4. Conclusion

Cutting-edge research has evidenced that RRT and RRT* based approaches are successful for high dimensional complex problems. The analysis of these approaches have shown that they confront issues of optimality and large number of nodes in search space. They improve path quality at the cost of computational efficiency. This performance trade-off could lead to more difficulty while dealing with high dimensions, high degrees of freedom and non-holonomic constraints. We plan to grow search tree more intelligently with improved computational efficiency for high dimensional complex problem using wheeled mobile robots. Future work in this direction is a thriving area of research.

References

[1] X. Lan, and S. Di Cairano, "Continuous Curvature Path Planning for Autonomous Vehicle Maneuvers Using RRT*", presented at the European Control Conference (ECC), 2015.

[2] D. Alejo, J. A. Cobano, G. Heredia, J. R. Martínez-De Dios, and A. Ollero, "Efficient Trajectory Planning for WSN Data Collection with Multiple UAVs", in Cooperative Robots and Sensor Networks. vol. 604, ed: Springer International Publishing, 2015, pp. 53-75.

[3] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime Motion Planning using the RRT*", presented at

the IEEE International Conference on Robotics and Automation (ICRA) 2011.
[4] S. M. Lavalle, Planning Algorithms: Cambridge University Press, 2006.
[5] S. Karaman, and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning", The International Journal of Robotics Research, vol. 30, pp. 846-894, 2011.
[6] M. Elbanhawi, and M. Simic, "Sampling-Based Robot Motion Planning: A Review survey", IEEE Access, vol. 2, pp. 56-77, 2014.
[7] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime Motion Planning using the RRT∗", presented at the IEEE International Conference on Robotics and Automation (ICRA) 2011.
[8] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning", 1998.
[9] J. J. Kuffner, and S. M. Lavalle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning", in Proceedings of IEEE International Conference on Robotics and Automation, 2000, pp. 1-7.
[10] J. Nasir et al., "RRT*-SMART: A Rapid Convergence Implementation of RRT*", International Journal of Advanced Robotic Systems, vol. 10, pp. 1-12, 2013.
[11] Y. K. Hwang, "Gross Motion Planning-A Survey", ACM Computing Surveys, vol. 24, pp. 219-291, 1992.
[12] J. W. Loeve, "Finding Time-Optimal Trajectories for the Resonating Arm using the RRT* Algorithm", Master of Science, Faculty Mechanical, Maritime and Materials Engineering, Delft University of Technology, Delft, 2012.

Biographies

Iram Noreen received BSCS from The University of Lahore and MSCS from Lahore College for Women University, Lahore with distinction. Her research interests are image processing, computer graphics and robotics. Presently, she is associated with COMSATS Institute of Information Technology, Lahore as PhD scholar in Department of Computer Science.

Amna Khan completed her BSCS and MSCS from University of Engineering & Technology, Lahore. Currently, she is a Ph.D. scholar at COMSATS Institute of Information Technology, Lahore. Her research interest includes computer graphics, geometric modeling and robotics

Zulfiqar Habib earned his PhD degree in Computer Science in 2004 from Kagoshima University Japan followed by the award of Postdoctoral fellowship of two years by Japan Society for the Promotion of Science. Currently, he is holding position of full Professor at the Department of Computer Science, COMSATS Institute of Information Technology (CIIT), Lahore, Pakistan. He has established a laboratory for research in Vision, Image, Graphics (VIG), and Robotics at CIIT, and is working as the In-Charge of the Center for Research in VIG-Robotics founded by him in 2011. Habib has achieved various awards in education and research including Farogh-e-Taleem Gold Medal for the best performance in field of education, two Research Productivity Awards by Ministry of Science & Technology, Pakistan and three graduate merit fellowships by Japanese and German Governments. He is an associate editor of the International Open Transportation Journal, reviewer of many famous journals and is organizer of various international conferences. Habib's teaching & research interests broadly span the areas of Computer Graphics, including Computer Aided Geometric Design, Data Visualization, Geometric Modeling, Reverse Engineering of Images, Path Planning, Robotics, Computer Vision, and Digital Image Processing. He has published a large number of highly cited research papers in impact factor journals, supervising master and PhD students, and actively involved in funding projects.