Implementation of RRT* - Smart Towards Optimal Solution

Venkata Sai Ram Polina
A.James Clark School of Engineering
University of Maryland, College Park
sairamp@umd.edu

Denesh Nallur Narasimman
A. James Clark School of Engineering
University of Maryland, College Park
deneshn@umd.edu

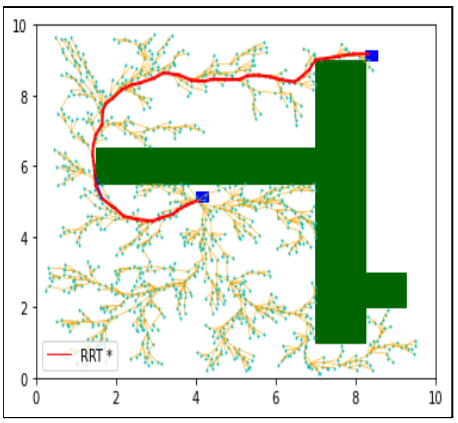
***Abstract*– In path planning, one of the most accurate, efficient and fastest algorithms to determine an obstacle free path is Rapidly Exploring Random Tree (RRT) but, it has its own limitations and hence, determining the optimal path is not assured. In order to overcome this limitation, Rapidly Exploring Random Tree Star (RRT*) was introduced recently which claims that it converges to get an optimal solution but it takes infinite time to converge. This limitation is overcome by an extension of the RRT* which is called the RRT* - Smart that is discussed in this report. The RRT* - Smart algorithm claims to converge to determine the optimal solution not only by increasing the rate of convergence but also by reducing the time required to execute the program. In order to achieve this, this algorithm consists of a couple of new techniques that are incorporated in the RRT* which are path optimization and intelligent sampling. The results and simulations shown in this report are a comparison between the above two mentioned algorithms; RRT* and RRT* - Smart.**

***Keywords*- Path Planning, RRT* algorithm, optimal solution.**

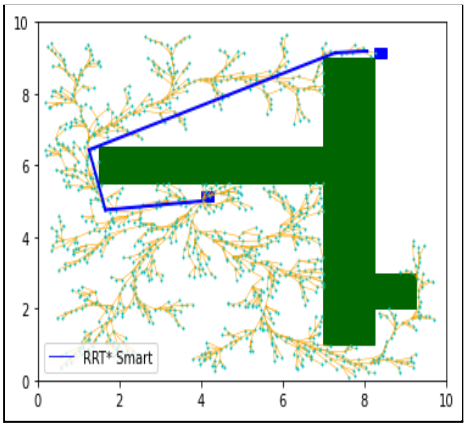
I. INTRODUCTION

The field of motion planning has attained a lot of importance and acceptance in recent years. It mainly concerns the determination of a feasible or economic path or trajectory to the goal node. This field has especially become more prominent in robotics as both the manufacturing and the domestic sectors are trending towards robotics and automation in a more efficient and intelligent manner.

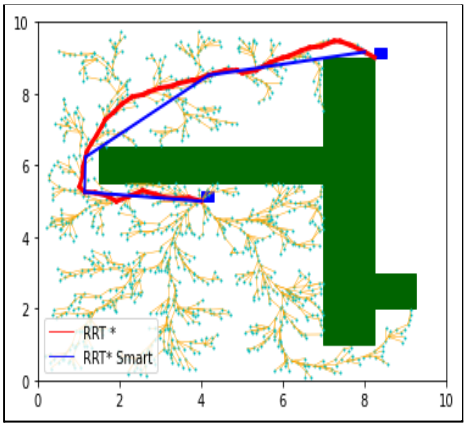
There were many algorithms that were introduced since the 1970's like the potential field algorithms, grid-based algorithms, neural networks and sampling based algorithms for both static and dynamic environments which also have their own advantages and disadvantages to find the most optimal path in terms of the space and time complexity and path optimization [1-6]. Among the above mentioned algorithms, sampling based algorithms are one of the latest and most used planning algorithms as they are computationally easier and they are capable of solving without any direct or obvious data of the obstacles in the environment as compared to the others. These algorithms depend upon a collision checking module and create a map of achievable paths obtained by linking the set of points that are sampled from the obstacle free environment. Rapidly Exploring Random Tree Star (RRT*) [7] is one such recently proposed algorithm which is based on sampling based algorithms and is popular for its rapid convergence rate to obtain the initial path. It then begins to optimize the initial path continuously as the sample size increases. But, the disadvantage of RRT* algorithm is, it never converges on the optimal solution in finite time and convergence rate is slow [7].



Initial path by RRT* with n=1200



Optimal path with RRT* - Smart with n=800



Comparison of optimal path between RRT* and RRT* - Smart

Fig 1. RRT* and RRT* - Smart optimal paths

This problem is solved by introducing RRT* - Smart, an improved version of RRT* which, with the help of its predecessor, determines the initial path and further explores the obstacle space. The final path or trajectory that is obtained with RRT* - Smart is straight and has less waypoints which makes it easy for the robot to track the path.

II. RRT* - Smart Algorithm

As discussed above, two new functions are incorporated here which are path optimization and intelligent sampling. The working of RRT* - Smart algorithm is same as the RRT* until it finds the initial path. This path is then optimized by easily linking the nodes that are explicitly visible. This process runs in a loop as the algorithm runs and the path is optimized continuously. Here, a biasing parameter is used to determine the shorter path when exploring and choosing that shorter path.

The *PathOptimization* technique basically, is an iterative process which begins from the goal node and moves towards the start node to see if there are any explicit nodes that are linked to their successive parent nodes until the condition that checks for collision fails. This technique is basically based on the concept of Triangular Inequality which states that *c* is always less than the sum of *a* and *b*. During this iterative process, at every step, a check for collision is done which is done using interpolation that does not need any data about the obstacles [8]. This implies that this method for checking the collisions does not depend upon the configuration of the obstacles in the environment. Additionally, this method is also computationally economical. By the end of the process of optimizing the path, the number of nodes will be very less when compared to the nodes when the initial path was determined, and these nodes are called *Beacons*. At this point, the beacons are then used as the basis for intelligent sampling.

Intelligent sampling on the other hand is the technique that is used here to achieve an optimal solution using the basis of visibility graph technique, in other words, generation of nodes as near to the vertices of the obstacles as possible.

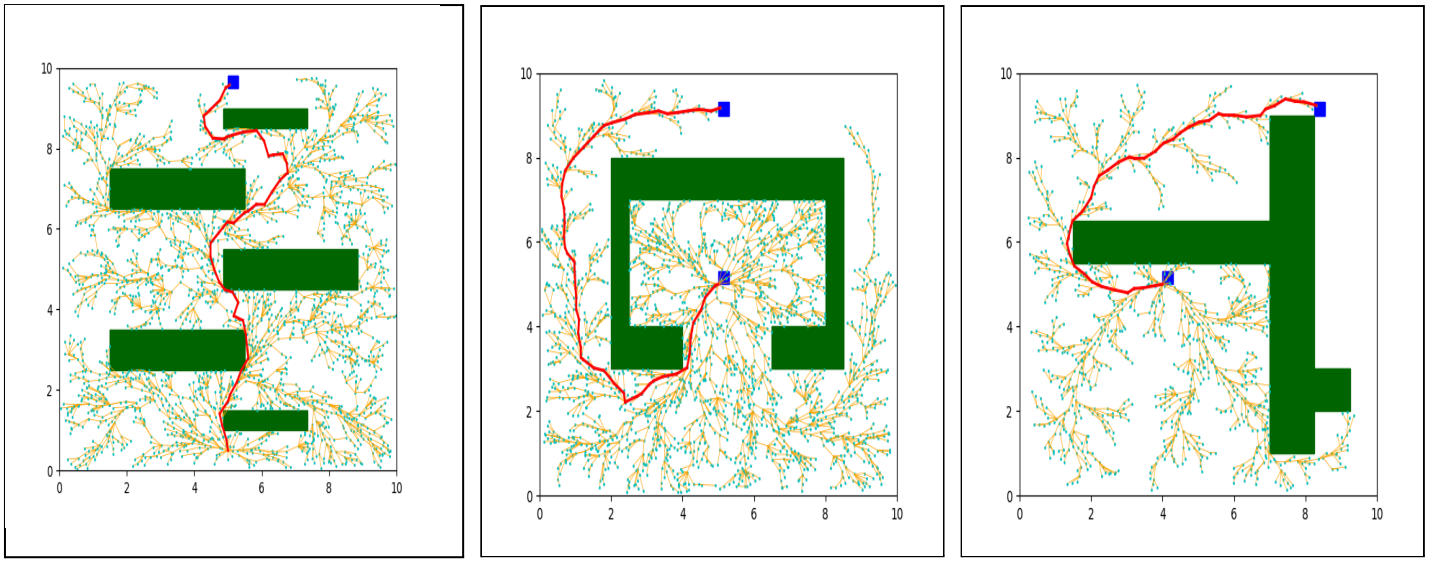


Fig 2. RRT* in different obstacle spaces

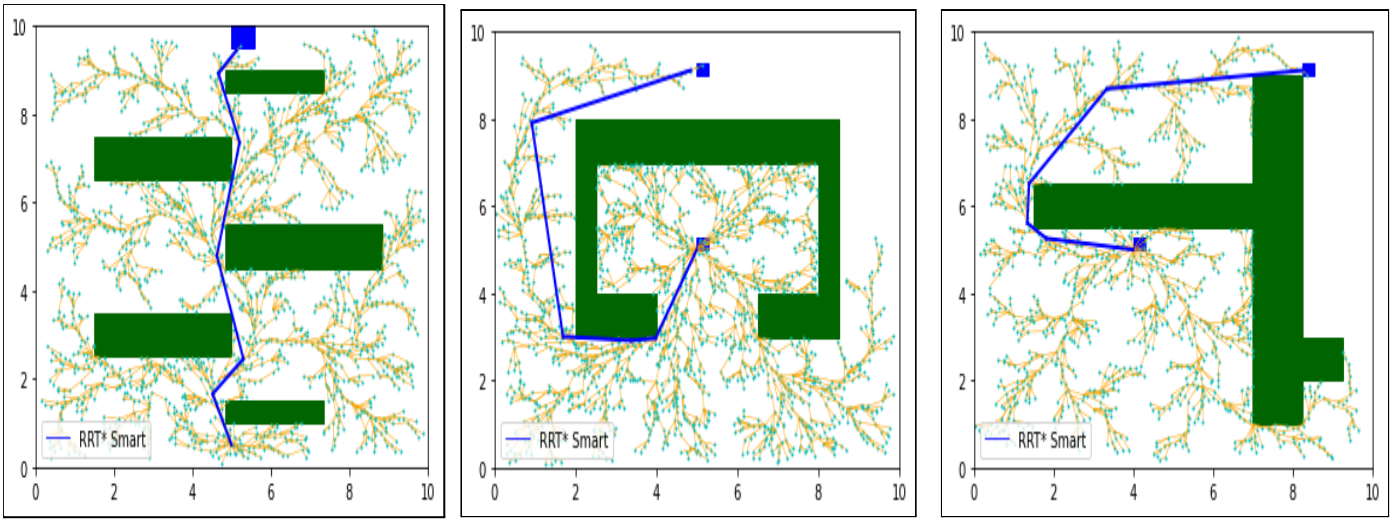


Fig 3. RRT* - Smart in different obstacle spaces

This needs complex environmental modeling and direct instructions and information of the obstacles in the environment [2]. However, this technique is only efficient in the case where the shapes of the obstacles do not have complicated shapes and geometries.

The role of intelligent sampling comes into picture once the initial path is found. It begins with selecting a set of samples that are present in a certain radius $R_{beacons}$ that has a center at $Z_{beacons}$. Priority is given to these beacons as these are the one's that help in giving the information about the edges and vertices of the obstacles (circumference in the case of circular objects). Hence, this implies that these beacons must be neighboring to a maximum number of nodes to get an optimized solution and therefore, the optimal solution is reached in lesser number of iterations when compared to RRT*.

As the algorithm runs, at each iteration, a new RRT* path is generated that has a lesser cost than the path generated previously with which an optimal path is found and the cost of the two paths are compared. Whenever the cost is less, new beacons; $Z_{beacons}$ are produced and thereby creating

Algorithm 2: $T= (V,E) \leftarrow$ **RRT*Smart**(z_{init})

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**

new optimized paths. This process runs in a loop until the required number of epochs are done.

Firstly, Let us understand the nomenclature used in the code.There are two main functions namely *RRTStarSmart* and *generateRRTStarTree*.

The algorithm execution starts from *generateRRTStarTree* function which calls the *RRTStarSmart* function.The RRT* tree is generated in the following way:

Sampling: A random node is generated using numpy uniform distribution and stored in to z_random

Nearest Node: A nearest node to generated node is found using *Nearest(z_random,T,Vertices)* where *T* stores information about each node and its coordinates while *Vertices* stores only the coordinates of nodes generated and the *Edges* stores coordinate of two connecting vertices. z_nearest (the nearest node) is returned

Split line: This dotted line represents the line connecting Z_nearest and z_random which is generated using *generatePath()* and assigned to new_path.

Collision check: The function *collisionWithRectangle()* checks if every sub-division of split line collides with an obstacle. If it does not collide a new node z_new is generated.

Nearest Neighbors: *Near()* returns nearest neighbors to the z_new generated and the cost and parent are assigned to New node z_new.

Rewire: The function *rewire()* checks if the cost of every node's parent has decreased or not.If the cost is decreased then z_near becomes a new parent and z_new becomes its child.

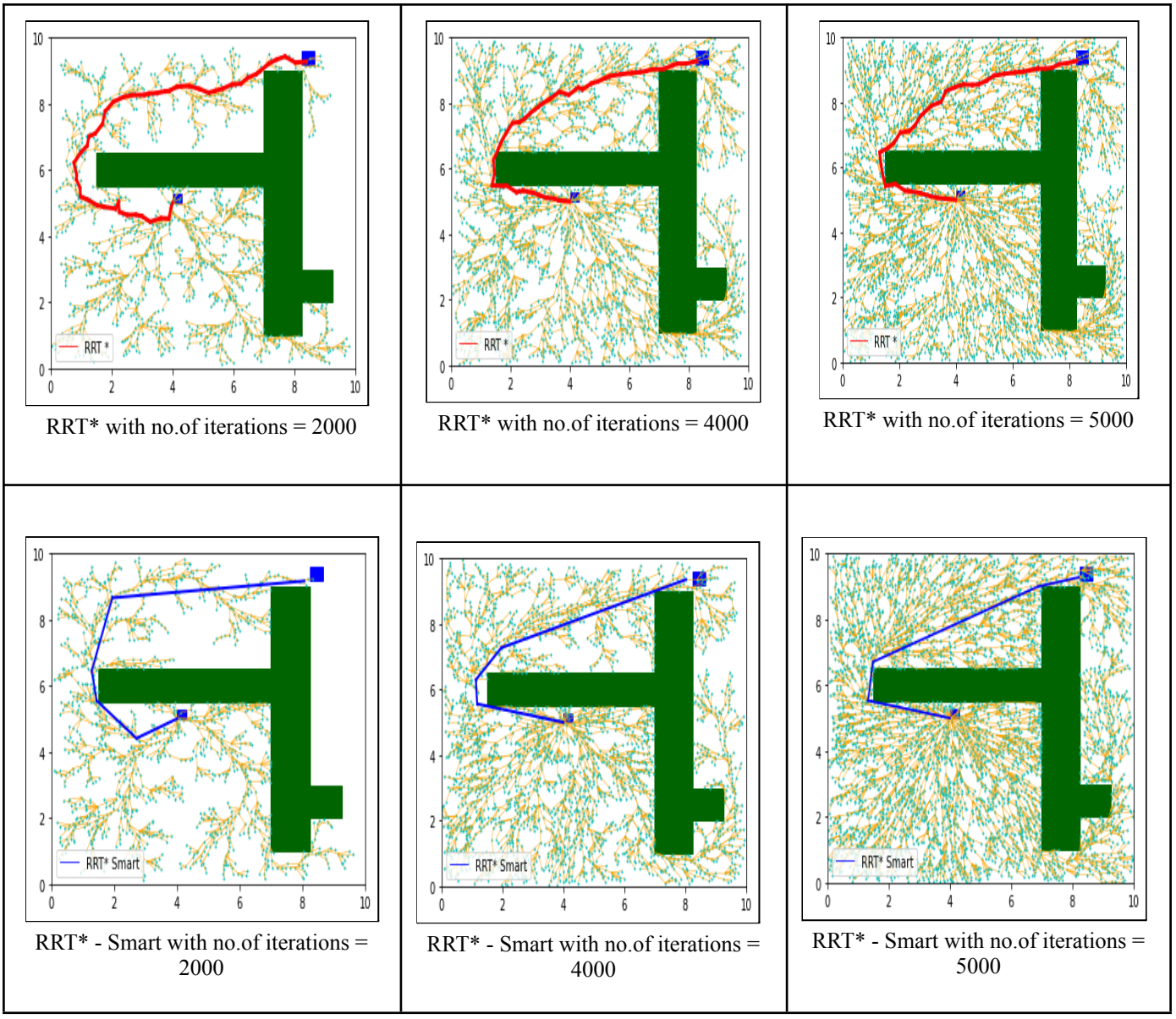
leastCost: The function *leastCost()* calculates the least cost between goal and start points.

Plot graph: The function *PlotTree()* plots the RRT* and RRT*Smart paths along with obstacles.

III. RESULTS

The results of RRT* and RRT* - Smart are compared in three different environments or obstacle spaces. The simulations show the optimal path achieved from the initial point to the goal point. The obstacles are represented in green color while the path is represented in red color and the nodes visited are represented in orange color. This shows that determining the optimal path using this path planning algorithm, does not depend on the shape or geometry of the obstacle as mentioned previously.

Comparison of experimental results between RRT* and RRT* - Smart are also discussed here considering various perspectives. Fig2 shows the determination of the optimal path using RRT*. For the first obstacle space, the optimal path was determined with a cost of 14.099 and number of iterations being 5000. The number of nodes explored for this case was 2440. For the second obstacle space, the optimal path was determined with a cost of 14.44 and number of iterations being 5000. The number of nodes explored for this case



RRT* with no.of iterations = 2000

RRT* with no.of iterations = 4000

RRT* with no.of iterations = 5000

RRT* - Smart with no.of iterations = 2000

RRT* - Smart with no.of iterations = 4000

RRT* - Smart with no.of iterations = 5000

Fig 4. Comparison between RRT* and RRT* - Smart at 2000, 4000 and 5000 iterations.

was 1900. For the third obstacle space, the optimal path was determined with a cost of 12.10 and number of iterations being 2300. The number of nodes explored for this case was 1309.

Fig 3 shows the optimal path that is determined using the RRT* - Smart algorithm in different obstacle spaces. Comparing figures 2 and 3, it is very clearly evident that the optimal path from the start node to the goal node by RRT* - Smart is achieved in a more efficient manner and by exploring a lesser number of nodes. For the first obstacle space, the optimal path was determined with a cost of 9.56 and number of iterations being 5000. The number of nodes explored for this case was 1800. For the second obstacle space, the optimal path was determined with a cost of 9.25 and number of iterations being 5000. The number of nodes explored for this case was 1500. For the third obstacle space, the optimal path was determined with a cost of 9.56 and number of iterations being 2300. The number of nodes explored for this case was 1250.

The figure 4 below, gives the pictorial representation of the optimal path that was determined using the RRT* and the RRT* - Smart algorithm at different iterations. We have chosen 2000, 4000 and 5000 to show the distinct differences between the two. From the figures, it is very clear that the optimal path is smoother for RRT* - Smart and is achieved by exploring lesser number of nodes when compared to RRT*. With the implementation of Intelligent sampling and path optimization. It is also evident that the rate of convergence of the RRT* - Smart is much faster than the RRT* and also it reaches the optimal cost within a finite number of iterations when compared to RRT* where it is still processing to reach an optimum result. It is also observed that, to reach the similar cost the RRT* algorithm takes more time when compared to the RRT* - Smart algorithm.

Upon considering and studying all the results, RRT* - Smart is efficient in terms of path optimization. Additionally, it showed very good results in terms of time of computation.

REFERENCES

[1] M. Kanehara, S. Kagami, J.J. Kuffner, S. Thompson, H. Mizoguhi, "Path shortening and smoothing of grid-based path planning with consideration of obstacles," IEEE International Conference on Systems, Man and Cybernetics, 2007 (ISIC) , pp.991-996, 7-10 Oct. 2007.

[2] I. Petrovic and M. Brezak, “A visibility graph based method for path planning in dynamic environments”, in proceedings of 34th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp.711-716,2011.

[3] N.H. Sleumer, N. Tschichold-German, “Exact cell decomposition of arrangements used for path planning in robotics" Technical report. Switzerland: Institute of Theoretical Computer Science Swiss Federal Institute of Technology Zurich; 1999.

[4] Y.K.Hwang, N. Ahuja , "A potential field approach to path planning," IEEE Transactions on Robotics and Automation,, vol.8, no. 1, pp.23-32, Feb 1992.

[5] A. Ghorbani, S, Shiry, and A. Nodehi,”Using Genetic Algorithm for a Mobile Robot Path Planning”, Proceedings of the 2009 International Conference on Future Computer and Communication ICFCC '09.

[6] S.X. Yang, C. Luo , "A neural network approach to complete coverage path planning," IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 34, no. 1, pp. 718- 724, Feb. 2004.

[7] S. Keraman and E. Ferazzoli, “Sampling-based Algorithms for Optimal Motion Planning” , International Journal of Robotics Research,2010.

[8] Bialkowski, Karaman, and Frazzoli, “Massively Parallelizing the RRT and the RRT*,” in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011.