Turtlebot Path Planning using Rapidly-Exploring Random Trees (RRT) and its Variants

Gourab Ghosh Roy*, Raabid Hussain** and Kibrom Berihu Girum***
VIBOT, Escola Politcnica Superior
Universitat de Girona
c/ Maria Aurlia Capmany, 61, 17071, Gerona, Spain
myself.gourab@gmail.com, raabid236@gmail.com, kibrom2b@gmail.com

Abstract— **This project presents different variations of the rapidly exploring random tree (RRT) strategies for optimal path planning solutions. The platforms used for the project were ROS for programming and gazebo for simulation. The robot is first tele-operated inside a static environment in order to build the map using gmapping techniques. The map is then used to plan the trajectory of the robot using different algorithms. RRT, RRT*, multiple RRT and RRT-connect were implemented. An approach to combine RRT with brushfire is also formulated. The primary focus was to reduce the computation time, distance of the trajectory and to adapt a trajectory that is as far as possible from the obstacles. The algorithms were tested on different environments available in gazebo, followed by a comparison between the techniques.**

I. INTRODUCTION

The main goal of an autonomous robot is to be able to construct and use a map and localize itself in it. Thus in autonomous robotics mapping and path planning are the main tasks of the robot and can be accomplished using a number of different algorithms depending on the application. Path planning is a well-known problem in robotics[6]. It can be defined as the determination of a path that the robot must take in order to pass over each point in an environment. The path is a plan of the geometric coordinates of the points in a given space which the robot has to pass through. Path planning algorithms try to generate an optimum path taking into account the obstacles in the environment. Although, motion planning is not the only fundamental problem in robotic architectures, perhaps it has gained popularity among researchers due to its widespread potential applications such as in robotics, assembly maintenance, computer animation, computer-aided surgery, manufacturing, and many other aspects of daily life [8-11].

Over-time, many path planning algorithms have been developed. Sampling based algorithms have gained popularity owing to the fact that if there is a feasible path, they are guaranteed to discover it. These algorithms were introduced to solve higher dimensional planning problems [12], which have the advantage of avoiding explicit construction of obstacle configuration spaces over other state-of-the-art algorithms. These algorithms ensure probabilistic completeness which implies always finding a solution, if one exists [13]. The sampled-based algorithms have proven to provide a computationally efficient [14] solution to motion planning problems. The most well-known sampling-based algorithms include Probabilistic Road Maps (PRM) [15, 16] and Rapidly exploring Random Trees (RRT) [17]. However, PRMs tend to be inefficient when obstacle geometry is not known beforehand [18]. Therefore, in order to derive efficient solutions for motion planning in practical world, the Rapidly-exploring Random Trees (RRT) algorithms [17] have been extensively explored. Various approaches have been used to enhance the original RRT algorithm [18-21]. The most remarkable variant of the RRT algorithm is RRT*, an algorithm which guarantees eventual convergence to an optimal path solution [18], unlike the original RRT algorithm. Hence the additional guarantee of optimality makes the RRT* algorithm very useful for real-time applications [22].

In this paper, we present our work involving different path planing algorithms. Turtlebot simulator was used to implement and compare different variants of the RRT algorithm: Rapidly exploring Random Tree(RRT), Multiple Rapidly exploring Random Tree (m-RRT), Rapidly exploring Random Tree Star (RRT*) and Rapidly exploring Random Tree Connect (RRT-Connect). The size of the robot was taken into consideration to avoid collisions with the obstacles. In order to further enhance the algorithms Brushfire Potential Fields (Brush-RRT) was incorporated into the architecture.

The rest of the paper is organized as follows: state of the art strategies, collision avoidance approaches, simulation specifics, experimental results and concluding remarks.

II. STATE OF THE ART

A. Rapidly exploring Random Tree (RRT) Algorithm

The RRT is a single query and probabilistically complete algorithm. The algorithm was originally proposed by LaValle in [18] and [19]. The algorithm starts by prompting an initial and a goal position. The goal is to form a tree structure in the configuration space from the initial position towards the goal position. A random point, q_rand, is generated in the free space in the map. It then selects the nearest neighbor from this q_rand and extends the tree from the nearest neighbor towards the q_rand, moving with a predefined step size given there is no obstacle in between as depicted in Fig. 1. If the two points are close together (smaller than the step size) then the tree is only extended up to the q_rand. The algorithm stops when the goal position is added to the tree.

Generation of the tree is followed by a smoothing step. The path form the initial position up to the goal position

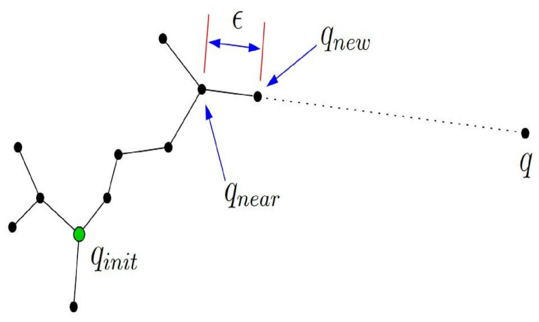

Fig. 1: Extending the tree using RRT algorithm.

can be found by back tracking the tree from its final node. However, this path is arbitrary with a lot of different movements (rotations/directions) for the robot which is not ideal. In order to smoothen and shorten the path, the goal and start positions are first connected to each other. If there is an obstacle in between, the connection is tried from a nearer node in the generated path. This is repeated for all the newly added nodes in the smoothened path until it reaches the starting position. RRT does not give the optimum path but its output can be optimized to some extent using this smoothing operation.

A certain amount of bias is added to the tree generation structure in order to speed up the process. A greedy approach is used to generate the q_rand by considering the goal point as the new q_rand. This guides the direction of the tree growth towards the goal position. An ideal case might be to always use the goal position as the q_rand however, the algorithm has to take care of obstacles and concave surfaces so the goal position is chosen as q_rand with a certain probability.

Algorithm 1 RRT (q_{init}, N)

```
1:  $V \leftarrow \{q_{init}\}$
2:  $E \leftarrow \phi$
3:  $T = (E, V)$
4:  for $i = 1$ to $N$ do
5:      Generate $q_{rand}$
6:      $q_{near} \leftarrow$ Nearest Neighbor $(T, q)$
7:      Get $q_{new}$ by moving $\epsilon$ from $q_{near}$ to $q$
8:      if path from $q_{near}$ to $q_{new}$ is free then
9:          $V \leftarrow V \cup \{q_{new}\}$
10:         $E \leftarrow E \cup \{(q_{near}, q_{new})\}$
11:     end if
12: end for
13: return $T$
```

B. Multiple Rapidly exploring Random Tree (mRRT) Algorithm

The idea of multiple RRT was introduced by Luna et al. in [20]. The notion followed here is that every time any probabilistic algorithm is run, it always gives a different answer. Similar is the case with RRT based algortihms. The generated tree is always different. A typical scenario in which RRT is run three times on the same map is shown in Fig. 4. So the same inputs are fed into the RRT algorithm, multiple times and their output are saved. When the robot has to move, the best solution among the different trials is opted. Typical definition of the best solution is the one with the shortest path but other criteria may be given preference too.

C. Rapidly exploring Random Tree Star(RRT*) Algorithm

RRT-star is the most popular modification of the original RRT algorithm. Here the main focus is on optimizing the path length. Proposed by Karaman and Frazzoli in [21], the algorithm aims at optimizing the tree structure at each iteration. Two additional steps are incorporated into the original structure. Firstly, after generating the q_rand, a neighborhood is opened around the point as shown in Fig. 3. Instead of connecting the new q_rand with the nearest neighbor, it is connected with the visible node in the neighborhood that gives the shortest distance from the start position to the new point/node.

Algorithm 2 RRT* Extend operation $1(T, q)$

```
1: $q_{near} \leftarrow$ Nearest Neighbor $(T, q)$
2: Get $q_{new}$ by moving $\epsilon$ from $q_{near}$ to $q$, $q_{min} \leftarrow q_{new}$
3: $Q_{near} \leftarrow$ Neighbors $(T, q_{new}, d)$
4: for $q_n$ in $Q_{near}$ do
5:     if Cost($q_n$) + dist($q_n, q_{new}$) $<$ Cost($q_{new}$) then
6:         $q_{min} \leftarrow q_n$
7:     end if
8: end for
9: $V \leftarrow V \cup \{q_{new}\}$, $E \leftarrow E \cup \{(q_{min}, q_{new})\}$
```

Secondly, all the visible points in the neighborhood are connected with the new q_rand. The distance to each such node in the original tree is compared to the distance with the new configuration. If the original distance is shorter, no change to the tree is made, otherwise the original connection to the neighbor node is removed and exchanged with the new path. In a tree structure it is important that each node should have only one parent although it can have many children/sub-branches.

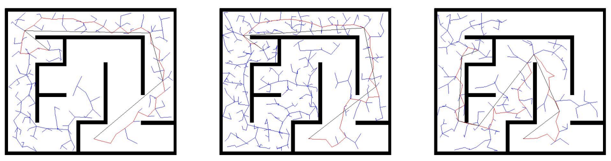

Fig. 2: Obtaining different trees through m-RRT algorithm.

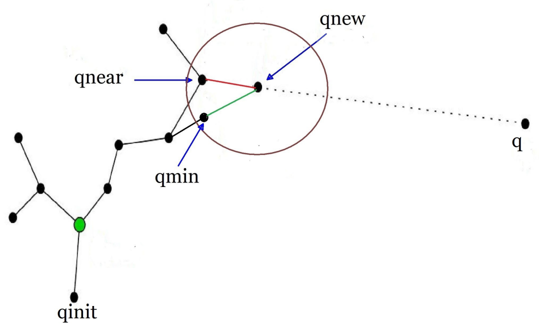

Fig. 3: Extending the tree using RRT* algorithm.

Algorithm 3 RRT* Extend operation 2(T, q)

```
10: for $q_n$ in $Q_{near} \backslash q_{min}$ do
11:     if Cost($q_{new}$) + dist($q_{new}, q_n$) $<$ Cost($q_n$) then
12:         $E \leftarrow E \backslash \{(Parent(q_n), q_n)\}, E \leftarrow E \cup \{(q_{new}, q_n)\}$
13:     end if
14: end for
15: return $T$
```

These two operations optimize the path length of the tree. The main difference in appearance between the RRT and RRT* trees is that RRT tree tends to have shorter branches spreading in random directions whereas the RRT* tree tends to have more organized branches spreading mostly towards same half of the plane.

D. Rapidly exploring Random Tree Connect (RRT-Connect) Algorithm

Unlike the RRT* and m-RRT algorithms, the RRT-connect algorithm aims at optimizing the computational cost. The approach was proposed by Kuffner and LaValle in [22]. The main idea is that instead of growing just one tree, two separate trees are grown simultaneously, one originating at the starting position whilst the other at the goal position. As can be seen in Fig. 4, both trees grow towards each other until a common connection is found. Since two trees are grown together, the convergence to the solution is faster.

The algorithm starts by initializing a tree at the starting position and the other at the goal position. A random point is generated to expand the tree originating from the starting position following the customary RRT structure. The algorithm then tries to connect the current tree with the other tree. In case a connection is successful, the algorithm stops otherwise the same process is repeated, swapping the current tree after each iteration. This way both the trees grow together.

Algorithm 4 RRT-Connect Planner(q_{init}, q_{goal})

```
1: Initialize $T_{q_{init}}$, $T_{q_{goal}}$
2: for $i = 1$ to $N$ do
3:     Generate $q_{rand}$ , Extend RRT $(T_{q_{init}}, q_{rand})$
4:     if Connect RRT $(T_{q_{goal}}, q_{new})$ = Success then
5:         return PATH($T_{q_{init}}$, $T_{q_{goal}}$)
6:     end if
7:     SWAP ($T_{q_{init}}$, $T_{q_{goal}}$)
8: end for
```

In order to search for a connection, the algorithm extends the newly augmented node in the direction of the other tree. The extension takes place with fixed distance intervals. If an obstacle is encountered in between, the function returns failure.

Algorithm 5 Connect (T, q)

```
1: while No Collision do
2:     EXTEND($T, q$)
3: end while
```

III. Collision Avoidance

When the above-mentioned algorithms were implemented on the turtlebot simulator, a few problems were observed. When a map of the environment is created using the robot sensors, the obstacles and the walls were found to be displaced from their centers and smaller than their actual sizes. This is because the robot sensors only provide an approximation of the sizes and positions of the objects. Also, the planning algorithms consider the robot itself as a single point whereas in reality the turtlebot is circular with a radius of around 30 cm. When the trajectory was planned and executed, it was found that the robot passes very close to the obstacles and sometimes even collides with them. Since no non-odometry sensors are being used while moving, it is very important to avoid even the slightest of collisions. The following two measures were taken to improve the collision avoidance of the robot.

A. Size of the Obstacles

Most of the times, it was observed that the mapped obstacles in the environment were reduced in size. Also, the boundaries were not uniform and they had small gaps

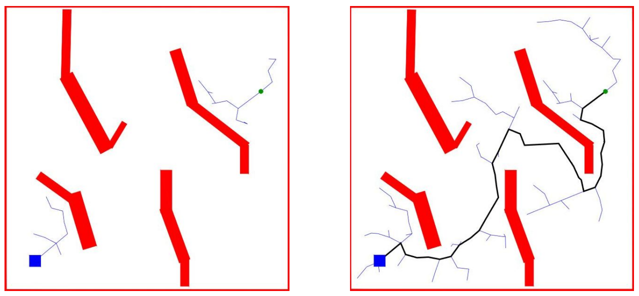

Fig. 4: Extending the trees using RRT-connect algorithm.

in them. A simple solution proposed is to increase the size of the obstacles using morphological operations. This had two advantages: the obstacles were better represented in the map and the size of the robot can be taken into account. An example of the effect of dilation is shown in Fig. 5.

B. Brushfire Potential Fields (Brush-RRT)

Another solution proposed for collision avoidance is to integrate potential fields into the RRT structure. Brushfire algorithm was chosen for this. It creates a potential map of the environment with respect to the distance from obstacles. The walls and obstacles in the environment are represented by higher potential values and as the position moves away from the obstacles, the potential decreases. The natural movement of any physical object is to go from a state of higher potential to a state of lower potential.

When a new q_rand is generated in RRT algorithm, a neighborhood is opened around the point as shown in Fig. 6. The generated q_rand is replaced by the location with the lowest potential in the neighborhood. As depicted in Fig. 6, the new branch of the tree is now further away from the obstacles thus decreasing the chances of collision.

As the path followed by the robot is not the one generated by the RRT algorithm rather a smoothed one extracted from the tree, so it is important to also include the brushfire into the smoothing framework. A direct path is only considered if all the points on it have a poential lower than a threshold.

IV. Methodology

The main task of the project was to implement path planning techniques on a turtlebot simulator based on ROS. Since different variants of the RRT algorithm were to be simulated, a map of the environment was required first. Two options were available. Firstly, pre-known maps of different environments could be used to test the algorithms. However, this limited the scope of the implementation. So, instead a turtlebot simulator called 'Gazebo' was used to first create a map of any arbitrary environment and then to plan the trajectory and move the robot.

Firstly, different environments were created in gazebo. To fully assess potential of the algorithms, map with different complexities were generated: ranging from simple maps with open spaces and few obstacles to complex maps with mazes, large number of obstacles and narrow spaces. Two sample environments are shown in Fig. 7. The environments were saved as '.world' files.

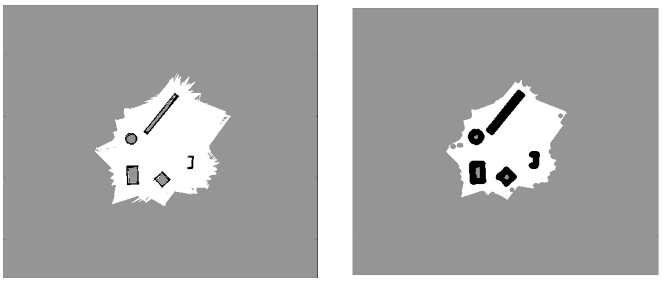
Fig. 5: Increasing the size of the obstacles using Dilation.

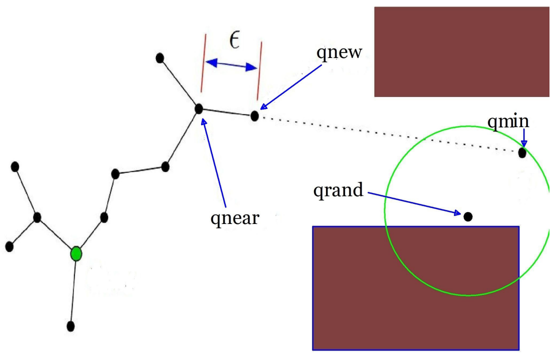

Fig. 6: Extending the tree using Brush-RRT.

These environments were then loaded back in gazebo and using 'gmapping' and tele-operation, a readable map of the environment was built. 'gmapping' using laser-based Simultaneous Localization and Mapping (SLAM) algorithm to generate the map. The 'gmapping' module does not provide the perfect replication of the environment, however it provides a satisfactory approximation of the environment. As the turtlebot was being teleoperated, the map being created was simultaneously observed using 'rviz navigation tools'. One such map is shown in Fig. 8. Obstacles are represented with high probabilities while the free spaces are represented with low probabilities in the map. After the map of the entire map has been created, the 'gmap' is saved as a 2D occupancy grid in form of a '.pgm' which is a raw image of the map and a '.yaml' file which contains information on how to read and utilize the map.

The gmaps were read using ROS map server to generate a thresholded 2D matrix of the environment. The typical values for the gmaps were 1 for free spaces, -1 for unexplored areas and 100 for obstacles. For simplicity the value of the unexplored areas was forced to 50. This occupancy grid was input to the path planning algorithms to generate a path based on the strategies mentioned in the previous sections. To reduce complexity during movement, only keypoints in the generated path were extracted from the path planning functions.

For movement of the robot, the waypoints output by the path planning algorithms were transformed back into the world coordinates (distances from pixels/matrix indexes). The movement of the turtlebot was controlled by the online odometry sensor feedback and the computed angular and linear velocities to go from one waypoint to the next. Since the environments were static and the issues of obstacle mis-alignments have been taken into account already, no obstacle detection sensors were required during the movement phase. For a smooth motion, a differential module was implemented for publishing velocity commands.

V. Experimental Results

After programming and testing all the three main stages (mapping, planning, motion) for a sampling based path planning algorithm, a detailed analysis was carried out to compare all the algorithms implemented. The main goal of an optimum path planning algorithm is to reduce the path length. However, in our experiments, three aspects were considered: path length, computational cost and collision avoidance.

The algorithms were tested on different environments, however for presenting two sample environments shown in Fig. 7 will be considered. The first environment is a rather simple and small with objects casually placed in an open space. Whereas the second environment is complex with different rooms, obstacles and narrow paths. These two environments have been chosen for the results discussion because both of them are different from each other and both resemble realistic situations that a robot might face. Since the algorithms are probabilistic, the results shown are an average of 5 trials.

The first attribute that is considered is the path length. Table I displays the path lengths for both environment using all four algorithms. RRT* gives the best path length owing ot the fact that it optimizes the tree by changing the entire tree structure during each iteration rather than just adding a new branch. At each iteration, it optimizes the path to each node in the q_rand's neighborhood depending to the path lengths. RRT-connect was slightly better than m-RRT because it uses two different trees that are biased towards each other. Owing to the nature of the connection finding phase, a more optimal path is sought. Whereas in m-RRT only the RRT simple algorithm is used multiple times and the shortest path among them is selected so behave better than simple RRT but falls short of RRT-connect.

The second attribute studied was the computational time. Table II displays the times in seconds for both environments using all the algorithms. RRT-connect was found to be most computationally inexpensive. This is due to the fact that two different trees are simultaneously grown towards each other. This helps to traverse though complex and concave areas more easily. RRT* and RRT performed better than the m-RRT because m-RRT runs the same algorithm many times before concluding which trial is the best.

The third attribute analyzed was the collision avoidance effect. The best way for this is to visually analyze the path generated. It was found that the final paths generated were passing close to the obstacle, most of the times. So the idea of Brush-RRT is introduced in this paper to solve this problem. The potential function was associated with all the four algorithms under study. Fig. 9a displays one output without using the brushfire algorithm for collision avoidance whereas Fig. 9b displays the output after considering the brushfire potential into the algorithm. As can be seen form the two figures, incorporating a potential method like brushfire is essential to obtain a path that is further away from the obstacle, thus reducing the chance of collision.

The detailed analysis revealed that RRT* gives the best output in terms of path length whereas the RRT-connect is least computationally expensive. Introducing potential function into the RRT structure reduces the chance of collision with the walls and the obstacles. It can be seen as essential because the when the robot is actually moving in an environemnt, there are many uncertainties involved that increase

TABLE I: Path length for different variants of RRT for same start and goal (averaged over 5 runs)

Environment	RRT	MRRT	RRT*	RRT-Connect
Environment 1	64.03	59.05	**56.35**	59.04
Environment 2	319.95	316.46	**295.523**	311.54

TABLE II: Planning times in seconds for different variants of RRT for same start and goal (averaged over 5 runs)

Environment	RRT	MRRT	RRT*	RRT-Connect
Environment 1	0.799	1.69	0.711	**0.615**
Environment 2	9.35	23.29	6.88	**5.71**

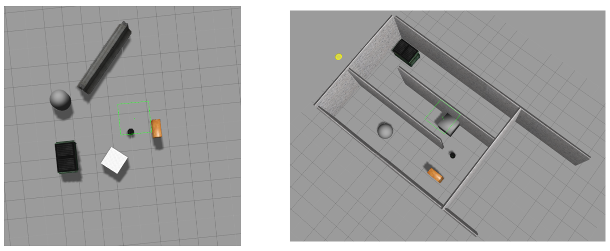
Fig. 7: Sample environments (Left: Simple environment with only obstacles; Right: Complex environment with walls and narrow paths).

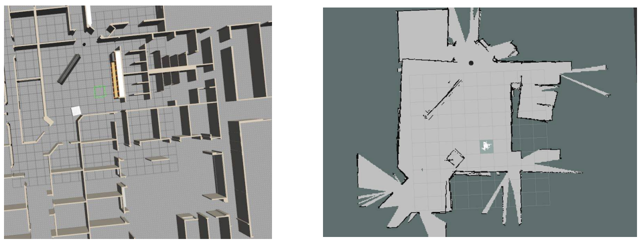
Fig. 8: The map generated of a sample environment using 'gmapping'.

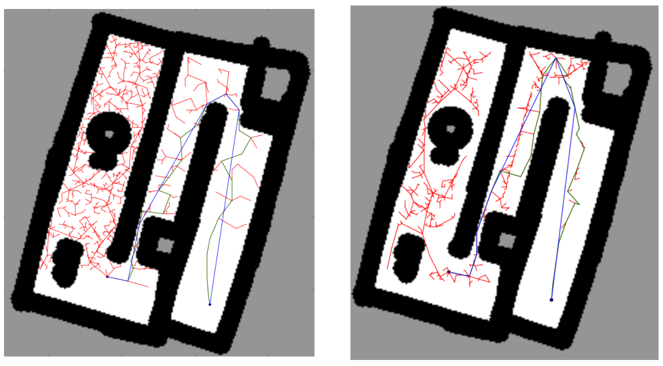
(a) RRT (b) Brushfire-RRT

Fig. 9: Output of the path planning algorithms.

the chance of collision.

VI. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

This project involved implementing various state of the art path planning strategies on a turtlebot simulator using ROS. Different variants of rapidly exploring random tree (RRT) structures, a popular branch of sampling based architectures, were implemented to obtain a collision free optimum path to the goal position. For enhanced performance, potential field obtained using brushfire algorithm was also integrated into the architecture for all the implemented algorithms. The in depth analysis revealed RRT-connect to be computationally superior whereas RRT* was found to provide the optimum path.

B. Future Works

This paper proposed a technique that used brushfire algorithm in RRT architecture to obtain a better solution in terms of distance from the obstacles. In order to further improve the motion of the robot, online measurements from the robot sensor may be used. Also, the presented approach is only suited for static environments. Different behavioral based techniques may also be introduced into the control architecture to deal with dynamic environments.

REFERENCES

[1] S.M. LaValle. Planning Algorithms, Cambridge University Press, 2006.

[2] J.-C. Latombe, ROBOT MOTION PLANNING,: Edition en Anglais, Springer, 1990.

[3] H. Change, T.-Y. Li, Assembly maintainability study with motion planning in: Robotics and Automation, 1995. Proceedings, 1995 IEEE International Conference on, Vol. 1,IEEE, 1995, pp. 1012-1019.

[4] M.Girard, A.A Maciejewski, Computational modeling for the computer animation of legged figures, in: ACM SIGGRAPH Computer Graphics, Vol. 19,ACM, 1985, pp. 263-270.

[5] R.D . Howe, Y.Matsuoka, Robotics for surgery, Annu. Rev .Biomed. Eng.1(1) (1999) 211-240.

[6] J,-C. Latombe, Motion planning: a journey of robots, molecules, digital actors, and other artifacts, int.J.Robots. REs.18(11)(1999) 1119-1128.

[7] M. Elbanhawi, M.Simic, Sampling-based robot motion planning: a review, IEEE Access 1(2014) 56-77.

[8] Ahmed Hussain Qureshi, Yasar Ayaz, Intelligent bidirectional rapidly-exploring random trees fro optimal motion planning in complex cluttered environments ,in: Robotics and autonomous systems68(2015)1-11:

[9] J.Canny. The compelxity of Robot Motion Planning. The MIT press, 1988.

[10] L Kavraki.J.-C. Latombe. Randomized preprocessing of configuration for fast Path planning, in: Robotics and Automation, 1994, Proceedings., 1994 IEEE International Conference on , IEEE. 1994.pp.2138-2145.

[11] L.E Kavraki.P.Svestka.J.-C. Ltome, M.H. Overmars, Probabilistic road maps for path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. 12(4)(1996) 566-580.

[12] S.M. LaValle. Rapidly-exploring random trees a new tool for path planning.

[13] S. Karaman. E. Frazzoli, Sampling-based algorithms for optimal motion planning. Int.J.Robot.Res.30(7)(2011)846-894.

[14] S.R.Lindermann. S.M. LaValle. Incrementally reducing dispression by increasing voronoi bias in RRTs. in: Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International conference on.Vol. 4.IEEE, 2004.pp.3251-3257.

[15] I.Garcia. J.P. How. Improving the efficiency of rapidly-exploring random trees using a potential function planner, in: Decision and contrl, 2005 and 2005 European Control Conference. CDC-ECC05. 44th IEEE Conference on, IEEE, 2005.pp.7965-7970.

[16] JJ. Kuffner Jr. S.M. LaValle. RRT-Connect: an efficient approach to single-query path planning, in: Robotics and Automation, 2000. Proceeding .ICRA'00. IEEE International Conference o. Vol. 2. IEEE, 2000.pp.995-1001.

[17] A.Perez, S. Karaman. A. Shkolnik, E. Frazzoli, S. Teller, M.R. Walter. Asymptotocally-optimal path planning for manipulation using incremental sampling-based algorithms, in: intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on IEEE. 2011.PP. 4307-4313.

[18] S. M. LaValle *Rapidly exploring random trees: A new tool for path planning*. TR 98-11, Computer Science Dept., Iowa State University, October 1998.

[19] S. M. LaValle and J. J. Kuffner. *Randomized Kinodynamic Planning*. International Journal of Robotics Research, 20(5):378400, May 2001.

[20] R. Luna, I. Sucan, M. Moll and L. Kavraki. *Anytime solution optimization for sampling-based motion planning*. In IEEE Int. Conf. Rob. Aut. , pages 5068-5074, 2013.

[21] S. Karaman and E. Frazzoli. *Incremental sampling-based algorithms for optimal motion planning*. In Robotics: Science and Systems (RSS), Zaragoza, Spain, June 2010.

[22] J. J. Kuffner and S. M. LaValle. *RRT-connect: An efficient approach to single-query path planning*. In Proceedings IEEE International Conference on Robotics and Automation, pages 9951001, 2000.

[23] B.J. Verwer, P.W. Verbeek and S.T. Dekker. *An Efficient Uniform Cost Algorithm Applied to Distance Transforms*. IEEE Transactions on Pattern Analysis and Machine Intelligence archive Volume 11 Issue 4 Page 425-429, April 1989.