Optimal Path Generation in 2D Map and Gazebo Environment Using A Sampling Based BTO-RRT Planning Algorithm

Anuj Zore
University of Maryland, College Park
zoreanuj@umd.edu

Vinay Krishna Bukka
University of Maryland, College Park
vinay06@umd.edu

***Abstract*- The author proposed an approach for searching obstacle-free, with low computational cost, and a smooth path with dynamic feasibility by analyzing the 2D maps of the target environment using a bi-directional and RRT Connect-based path planning algorithm. k-d tree-based strategy employs obstacle avoidance and three-step (Downsample, Upsample and Smooth) optimization. The author's approach is benchmarked with RRT, RRT and RRT Star.**

I. Introduction

For exploration, localization, mapping and navigation, mobile robots need to employ computationally efficient (optimal) and smarter (collision-free) path-planning strategies for performance optimization. Even though the RRT algorithm was fairly successful as one of the quickest and most efficient obstacle avoidance path-finding algorithms, it fails to perform similarly in narrow passes as it generates sharp edges (difficult to drive through).

The current research aims to compare and implement the results based on BTO-RRT(Bi Directional Target Oriented) from [1] which is an improved version from [2] written by the same author. Here, variants like RRT Connect[3], RRT *[4], and Bidirectional RRT*[5] come in - where they improved efficiency and rate of convergence. However, the algo doesn't consider the dynamic constraints of the vehicle. In CL-RRT[6], although they considered the UAV non-holonomic constraints and RRT-based controller design method are provided - the final path is not smooth enough and not cost-optimal.

The Bi Directional RRT algorithm simultaneously explores the configuration space in both directions by using two trees that grow from the initial and goal states. It assumes each other's starting point as the target. RRT-Connect, in contrast, iteratively connects two trees by growing them closer to one another. Two trees grow from initial and target point respectively with the end position of each other as the target, and connect to the last expansion of each other. This can be observed by Figure 1 and Figure 2. By changing the steering mechanism of the algorithm, BTO-RRT can be expanded to include both Bi-RRT and RRT-Connect. With bi-directional RRT and RRT connect - our approach becomes more target-oriented generating a zig-zag trajectory. Followed by 3 Stage optimization to shorten and smoothen the total distance of the zig-zag path. Additionally, the B- Spline curve (combination of intermediate point interpolation and cubic spline curve) ensures continuity in a clustered environment.

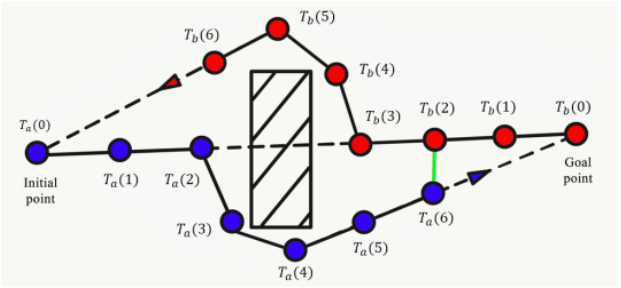

Figure 1: Bidirectional RRT core algorithm (Source : [1])

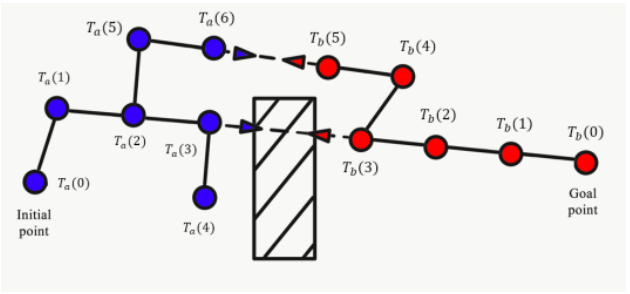

Figure 2: RRT- Connect algorithm (Source : [1])

II. METHOD

Proposed algorithm has two parts:

A. BTO-RRT Core

The overall flowchart for the BTO-RRT algorithm followed is in Figure 4. This part is responsible for a more target-oriented initial solution. BTO-RRT takes advantage of bi-directional RRT and RRT connect - modify the extension rules for more target-oriented with faster connections.

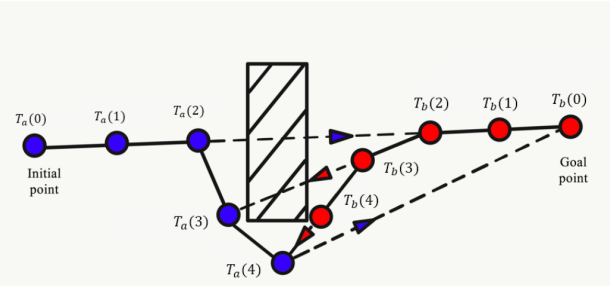

Figure 3: BTO-RRT (Source : [1])

In the proposed algorithm, Ta and Tb trees are defined with E, X-init and X-goal as the Edge of the tree, initial point and goal point respectively. In the above figure, tree

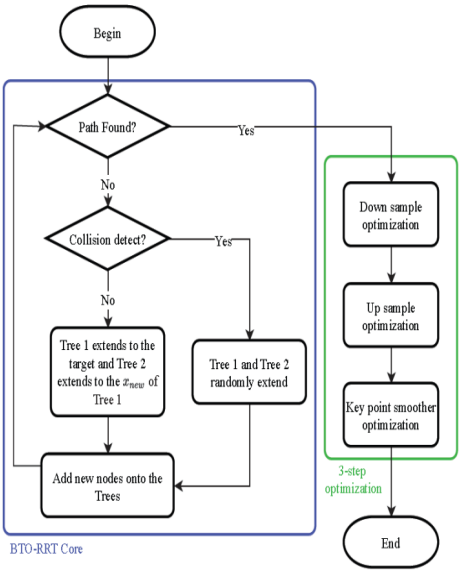

Figure 4: BTO-RRT (Source : [1])

Ta starting from the initial point always considers the goal point as the destination/goal point. Whereas the tree Ta expanding from Tb always considers the recently added node as the target destination point while taking fewer nodes to connect two trees. The pseudo-code is shown in the below figure:

Algorithm 1 BTO-RRT(x_{init}, x_{goal})

T_a.init(x_{init})
T_b.init(x_{goal})
$Pathfound \leftarrow False$
$i \leftarrow 0$
while $Pathfound \neq True$ or $i < N$ **do**
  $\{T_a, Flag_a\} \leftarrow$ EXTEND(T_a, x_{goal})
  $\{T_b, Flag_b\} \leftarrow$ EXTEND$(T_b, T_a(end))$
  $Pathfound \leftarrow Pathfound + Flag_b$
end while
$Path_a \leftarrow$ FINDMINIMUMPATH(T_a)
$Path_b \leftarrow$ FINDMINIMUMPATH(T_b)
$Path \leftarrow$ CONCATENATE$(Path_a, Path_b)$
return $Path$

Algorithm 2 EXTEND(T, x)

FLAG $\leftarrow 0$; $i \leftarrow 0$;
if COLLISIONDETECT = NOCOLLISION **then**
  $x_{rand} \leftarrow x$
else
  $x_{rand} \leftarrow$ Sample(i)
end if
$x_{nearest} \leftarrow$ NEARESTNODE(T, x)
$x_{new} \leftarrow$ STEER $(x_{nearest}, x)$
if COLLISIONDETECT = NOCOLLISION **then**
  $V \leftarrow \{x_{new}\}$; $E \leftarrow \{x_{new}, x_{near}\}$
  T $\leftarrow$ {V, E}
  if $|x_{new} - x| < StepSize$ **then**
    return FLAG = 1
  end if
end if
return T, FLAG

B. *Optimization Algorithm*

As the path generated from the BTO-RRT core is zigzagged further optimization is required for making the generated path more optimal and dynamically feasible. This part improves the initial path (generated from BTO-RRT) with 3-step optimization i.e. Down-sample optimization, Up-sample optimization and Key point smoother optimization.

- Down-sampled optimization
- Up-sampled optimization
- Key point Optimization

Down-sample optimization: This optimization part looks for collision status starting from the initial point followed by the connection tree nodes. It continues to look for a collision between the current point and its next node point - after finding one it marks the previous checking node point as the current breakpoint and connects it with the previous breakpoint. Effectively leading to the downsampling of the previously generated path. The downsampling is illustrated in the below-given diagram.

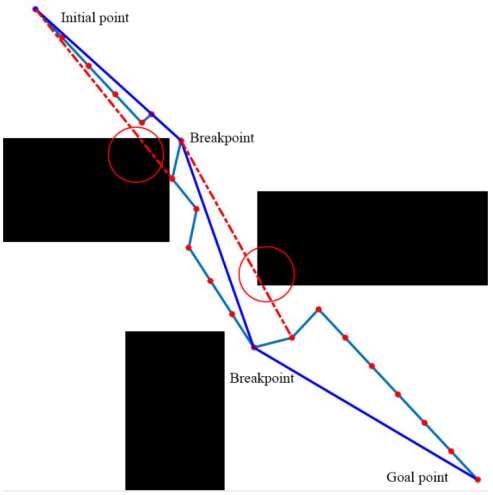

Figure 5: Breakpoints (Source : [1])

Up-Sample optimization: Even though after downsampling the corrected path is locally shorter, it's not close to obstacles, especially at the corners. Here, the up-sample helps in adding more sample points in between closer to the nearest obstacles by reducing the resulting distance globally.

Assume DSTn =

$$\{p_1^{DS}, p_2^{DS}, \ldots, p_n^{DS}\},\ UST_m = \{p_1^{US}, p_2^{US}, \ldots, p_m^{US}\}$$

For the initial iteration consider DST = UST where Cs, Csi are the cumulative sum of downsampled points. Each element is in the form of the next step involves taking two random distances based on the Euclidean distance between the start point and end point:

$$ranD_{1,2} = x \cdot Cs_{i=n}, x \sim U(0,1)$$

when it satisfies the condition: ranD1 <= ranD2. Followed by the step involved in inserting random interpolation

gamma1 and gamma2 into the region between Pi and Pj+1 where satisfies:

$$Cs_i = \sum_{i=1}^{i} p_i^{DS}, \forall i = 1,2,...,n$$

$$C_{s_i} \leq randD_1 < C_{s_{i+1}}$$

and j satisfies the inequality in

$$C_{s_j} \leq randD_2 < C_{s_{j+1}}$$

and gamma1 and gamma 2 are defined in the below equations:

$$a_1 = \frac{ranD_1 - Cs_i}{Cs_{i+1} - Cs_i}$$

$$a_2 = \frac{ranD_2 - Cs_j}{Cs_{j+1} - Cs_j}$$

$$\gamma_1 = (1 - a_1) \times p_i^{US} + a_1 \times p_i^{US}$$

$$\gamma_2 = (1 - a_2) \times p_i^{US} + a_2 \times p_i^{US}$$

which ensures that the connection path is collision-free. So, the new up sample points UST becomes:

$$UST_m^{'} = \{p_1^{US}, ..., p_i^{US}, \gamma_1, \gamma_2, p_{j+1}^{US}, ..., p_m^{US}\}$$

we repeat the above process until it reaches the maximum iteration (User defined in execution) - in each iteration, we insert globally shorter vertices to the path. It observed that with increased iterations the path becomes globally shorter. Below is the output of the upsampling on 500, 1000 and 10000 iterations respectively. The figures 6, 7, 8 show the optimized path via up-sample 500, 1000 and 10000 iterations respectively. Since the considered 2D map is of small size, there is only a small significant difference of generating optimal path after different iterations. It can be observed that our result after upsampling and parameter tuning is required - results can be improved increasing the number of iterations in upsampling section.

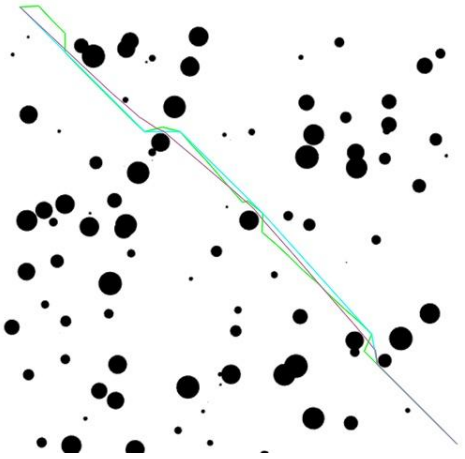

Figure 6 : Up-Sample Path(Dark Brown Line)

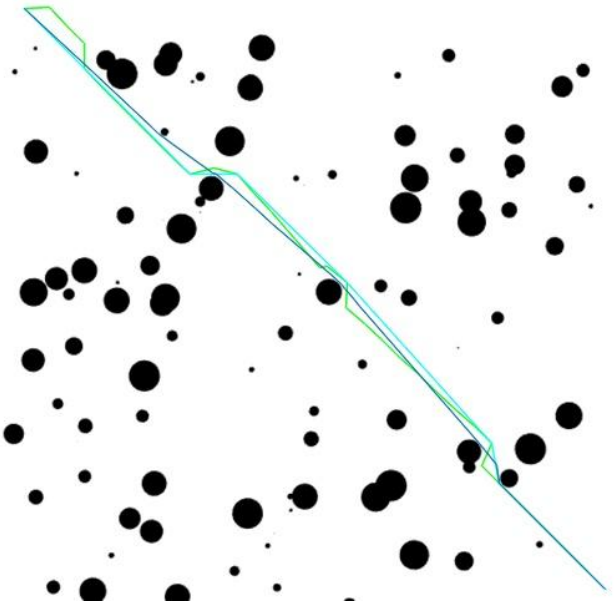

Figure 7 : Up-Sample Path(Dark Blue Line)

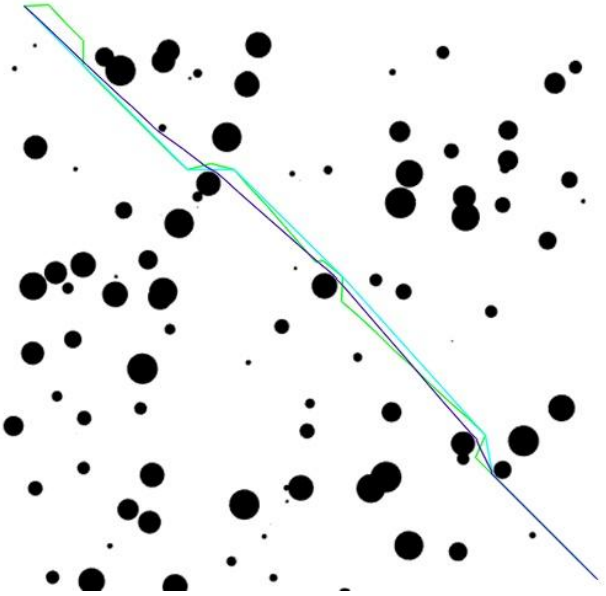

Figure 8 : Up-Sample Path(Purple Line)

III . **RESULTS**

The BTO-RRT Algorithm is implemented in the Python programming language. The algorithm has been tested on different maps. The explored path, BTO-RRT path, down-sample path and up-sample path are plotted in OpenCV. Each map is tested with a different number of iterations, step-size and distance threshold. In Each of the maps below red and blue lines represent the exploration of start and goal nodes. The green line represents the connected tree path once two trees get connected to each other after reaching a certain distance threshold. The light blue line represents the path generated after down-sampling, whereas the brown line represents the path generated by up-sampling. Also, table 1 , shows the

final path lengths for BTO-RRT core, up-sample and down-sample for map 5.

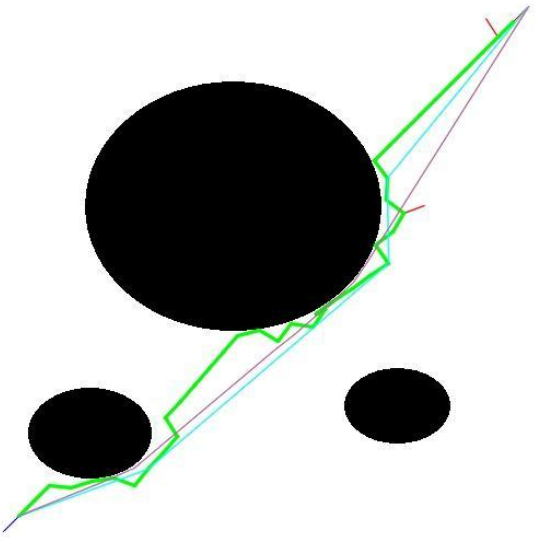

Figure 9 : Map 1

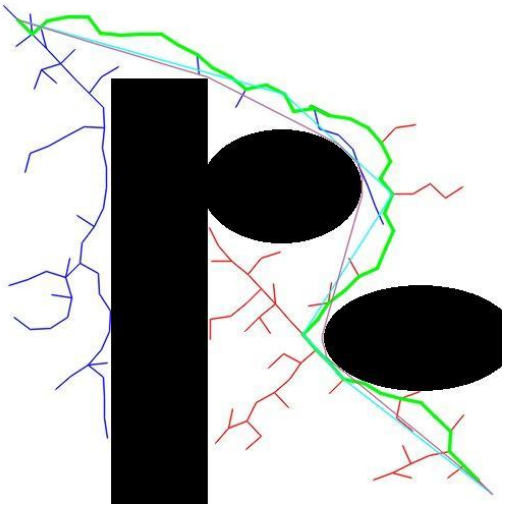

Figure 10 : Map 2

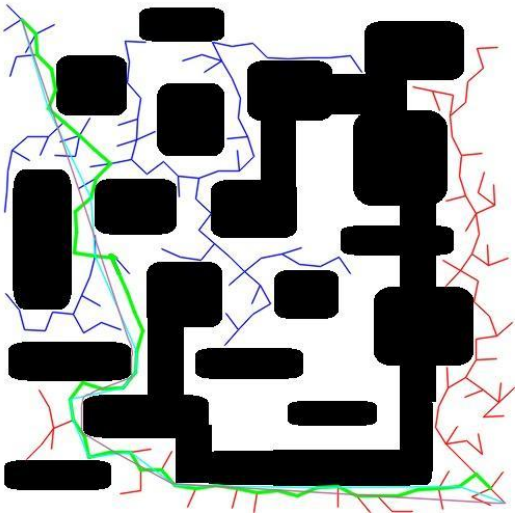

Figure 11 : Map 3

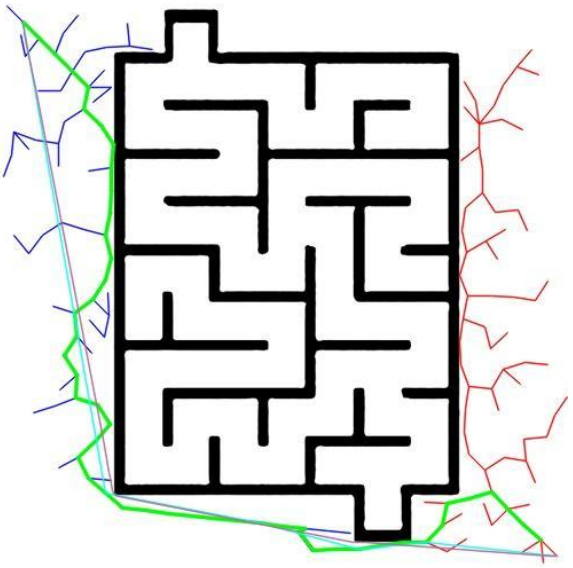

Figure 12 : Map 4

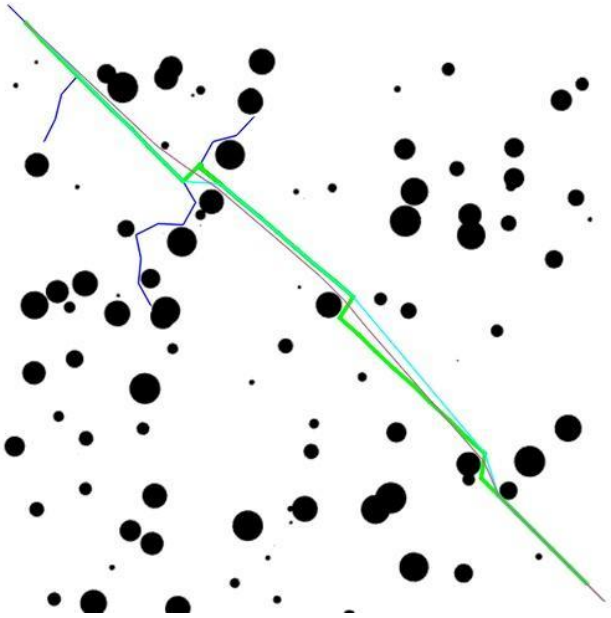

Figure 13 : Map 5

TABLE 1:

Method	Path-Length
BTO-RRT-Core	686.2127
BTO-RRT-DownSample	676.3153
BTO-RRT-UpSample-500-Iterations	664.1881
BTO-RRT-UpSample-1000-Iterations	663.7593
BTO-RRT-UpSample-10000-Iterations	663.7137

IV. SIMULATION - GAZEBO

Through this research we have applied the algorithm to the 3D environment Gazebo using ROS Noetic. The 3D implementation of Gazebo is not performed in the author's original research. A closed loop controller is implemented to control the robot's linear speed and angular speed while it is navigating through an optimal path and avoiding obstacles. The robot radius is added as clearance onto the obstacles. The Final Position of the robot once it reaches the goal position is shown in below 2D and 3D world figures.

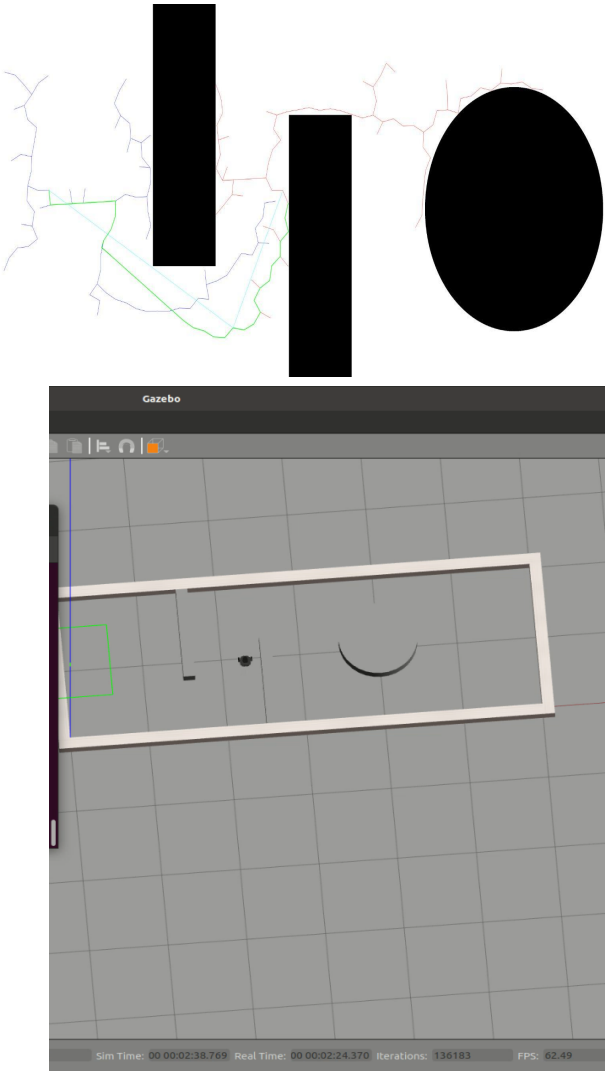

Figure 14 : 2D and 3D exploration Figures

V. CONCLUSION

The discussed algorithm is able to rapidly search for optimal, smooth and dynamically feasible trajectories both in 2D and 3D maps.. The 3-stage optimization allows this algorithm to globally shortest smooth path - and can be successfully implemented on point cloud maps. These optimizations stages can be applied to other sampling based algorithms improving the dynamically feasible and optimal. The future work planned is to improve the efficiency of navigation in the 3D environment Gazebo and also implement the algorithm in other 3D environments such as point clouds.

VI . REFERENCES

[1] Zheng, Z., Bewley, T. R., Kuester, F., & Ma, J. (2022). BTO-RRT: A rapid, optimal, smooth and point cloud-based path planning algorithm. arXiv preprint arXiv:2211.06801.

[2] Zhaoliang Zheng, T.R. BEWLEY, and Falko Kuester. Point cloud-based target-oriented 3d path planning for uavs. In 2020 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, 2020.

[3] Kuffner, James J., and Steven M. LaValle. "RRT-connect: An efficient approach to single-query path planning." Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065). Vol. 2. IEEE, 2000.

[4] Islam, Fahad, et al. "Rrt*-smart: Rapid convergence implementation of rrt* towards optimal solution." 2012 IEEE international conference on mechatronics and automation. IEEE, 2012.

[5] Jordan, Matthew, and Alejandro Perez. "Optimal bidirectional rapidly-exploring random trees." (2013).

[6] Naderi, Kourosh, Joose Rajamäki, and Perttu Hämäläinen. "RT-RRT* a real-time path planning algorithm based on RRT." Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games. 2015.

[7] Iram Noreen, Amna Khan, and Zulfiqar Habib. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. International Journal of Computer Science and Network Security (IJCSNS), 16(10):20, 2016.

[8] Christoph Sprunk. Planning motion trajectories for mobile robots using splines. 2008.

[9] Radu Bogdan Rusu, Nico Blodow, Zoltan Marton, Alina Soos, and Michael Beetz. Towards 3d object maps for autonomous household robots. In 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3191–3198. IEEE, 2007

[10] Shrihari Vasudevan, Stefan Gachter, Viet Nguyen, and Roland Sieg- ¨wart. Cognitive maps for mobile robots—an object based approach.Robotics and Autonomous Systems, 55(5):359–371, 2007.

[11] Hemant M Kakde. Range searching using kd tree. from the citeseerx database on the World Wide Web: http://citeseerx. ist. psu.edu/viewdoc/summary, 2005.

[12] Matthew Jordan and Alejandro Perez. Optimal bidirectional rapidly-exploring random