

PATH PLANNING USING GOAL BIASED GAUSSIAN DISTRIBUTION BASED RRT\*

A Thesis

by

PARTH JOSHI

Submitted to the College of Graduate Studies  
Texas A&M University-Kingsville  
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2018

Major Subject: Electrical Engineering

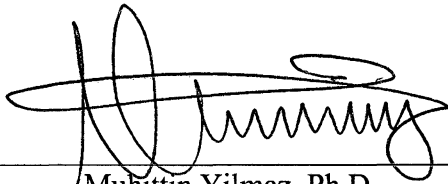
# PATH PLANNING USING GOAL BIASED GAUSSIAN DISTRIBUTION BASED RRT\*

A Thesis

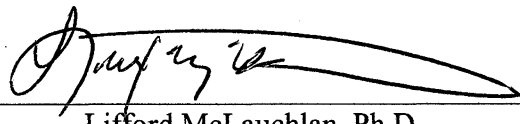
by

PARTH JOSHI

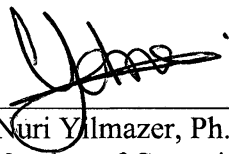
Approved as to style and content by:



Muhittin Yilmaz, Ph.D.  
(Chair of Committee)



Lifford McLauchlan, Ph.D.  
(Member of Committee)



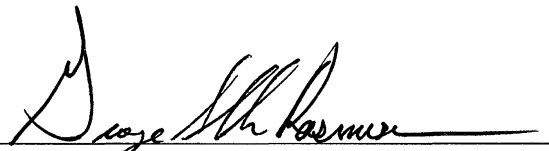
Nuri Yilmazer, Ph.D.  
(Member of Committee)



Selahattin Ozcelik, Ph.D.  
(Member of Committee)



Rajab Chaloo, Ph.D., P.E  
(Chair of Department)



George Allen Rasmussen, Ph.D.  
(Vice President for Research and Graduate  
Studies)

December 2018

## ABSTRACT

Path Planning Using Goal Biased Gaussian Distribution Based Rrt\*

(December 2018)

Parth Joshi, B.E. Electronics & Communications, GTU, Gujarat, India

Chair of Advisory Committee: Dr. Muhittin Yilmaz

Rapidly Exploring Random Tree (RRT) is a sampling-based heuristic path planning approach used. An extended version of RRT, i.e., optimal RRT (RRT\*), is widely used for path planning due to its asymptotic convergence and single query properties to find the optimal path for autonomous vehicles and robots. As contemporary autonomous vehicles demand more accurate and fast path planning algorithms, further improvements of RRT\* are needed for safe and superior operations. This research introduces a new approach in the RRT\* algorithm to find an optimal path in less computational time without compromising the useful characteristics of the RRT\*. The proposed algorithm uses the Goal Biasing methodology that focuses on tree expansion towards the end point and Gaussian Distribution, which helps to reduce the overall state space. The proposed algorithm is a modification in RRT\* for implementation of the goal biasing and Gaussian distribution methodologies. The proposed algorithm was evaluated on five different illustrative environments with different obstacle characteristics for its efficiency. The numerical experiments demonstrate that the proposed algorithm is more efficient regarding the distance cost for the path and computational time, when compared to RRT\*.

## **ACKNOWLEDGMENTS**

I would like to thank God for giving me strength, knowledge and blessings. I would like to thank my advisor Dr. Muhittin Yilmaz for his continuous support, motivation, help throughout my research, taking personal interest in my work and direct me during my hard times. I also like to thank my committee members Dr. Liford Mclauchlan, Dr. Nuri Yilmazer, and Dr. Selahattin Ozcelik for their opinion and support during my research. I also like to thank my parents and roommates for their unconditional and continuous support.

## TABLE OF CONTENTS

	Page
<b>ABSTRACT.....</b>	<b>iii</b>
<b>ACKNOWLEDGMENTS .....</b>	<b>iv</b>
<b>TABLE OF CONTENTS .....</b>	<b>v</b>
<b>LIST OF FIGURES .....</b>	<b>vii</b>
<b>LIST OF ALGORITHMS.....</b>	<b>ix</b>
<b>CHAPTER 1 INTRODUCTION.....</b>	<b>1</b>
1.1 Path Planning and Trajectory Planning.....	1
1.2 Literature Review .....	2
1.3 Motivation .....	6
<b>CHAPTER 2 BACKGROUND.....</b>	<b>8</b>
2.1 Problem Formulation.....	8
2.2 Rapidly Exploring Random Tree (RRT) Algorithm .....	9
2.3 1 <sup>st</sup> Layer of Optimization: Rewiring for Low-Cost Tree .....	17
2.4 2 <sup>nd</sup> Layer of Optimization .....	20
2.5 Optimal RRT (RRT*) Algorithm.....	21
<b>CHAPTER 3 PROPOSED METHODOLOGY .....</b>	<b>26</b>
3.1 3 <sup>rd</sup> Layer of Optimization: Goal Biasing.....	26
3.2 4 <sup>th</sup> Layer of Optimization: Gaussian Distribution .....	27

3.3 Proposed Goal Biased – Gaussian RRT* Algorithm .....	28
<b>CHAPTER 4 RESULTS.....</b>	<b>33</b>
4.1 Map-1: No Obstacle .....	37
4.2 Map-2: Width Effect .....	41
4.3 Map-3: Gap Effect.....	44
4.4 Map-4: Two Obstacle for Trap Check .....	48
4.5 Map-5: T Type Obstacle .....	50
<b>CHAPTER 5 CONCLUSIONS.....</b>	<b>53</b>
<b>REFERENCES.....</b>	<b>54</b>
<b>APPENDIX I .....</b>	<b>58</b>
7.1 MATLAB Code.....	58
<b>VITA.....</b>	<b>77</b>

## LIST OF FIGURES

	Page
Fig. 1 Elements of Node Based Optimal Algorithms [13].....	3
Fig. 2 Elements of Sampling-Based Algorithms [13].....	5
Fig. 3 Grid Check Method for RRT And A* .....	7
Fig. 4 Grid Check Method for RRT and Continuous Method .....	7
Fig. 5 Plane Subdivision and The Corresponding Tree Schematics [27] .....	11
Fig. 6 Steering Function [20] .....	11
Fig. 7 Tree Expansion of RRT From the Start Point .....	14
Fig. 8 New Point is Sampled.....	14
Fig. 9 Discovery of the Nearest Neighbor from the Vertex Set.....	15
Fig. 10 Creation of a Connection Between Two Points.....	16
Fig. 11 The Relation Between RRT and Voronoi Region [28] .....	17
Fig. 12 Discover a Vertex Point Within $r$ Radius .....	18
Fig. 13 Rewiring Process for New Tree.....	19
Fig. 14 Discover the Parent Nodes In the $r$ Radius .....	20
Fig. 15 Update the Parent Node for the Low-Cost Tree .....	21
Fig. 16 RRT* Tree Expansion (a) 100 Iterations, (b) 500 Iterations, (c) 2000 Iterations, (d) 5000 Iterations .....	25
Fig. 17 Gaussian Distribution Sampling [32] .....	28
Fig. 18 RRT* Implementation at Different Stages .....	34
Fig. 19 The Proposed GB Gaussian RRT* Implementation at Different Stages .....	35
Fig. 20 (a) Map-1 (b) Map-2 (c) Map-3 (d) Map 4 (e) Map 5 .....	36
Fig. 21 Map-1 Solution Using RRT*.....	38

Fig. 22 Map-1 Solution Using the Proposed GB Gaussian RRT*	38
Fig. 23 Comparison Between GB Gaussian RRT*, RRT* and Ideal Path	39
Fig. 24 Comparison Between Different Height to Distance Ratio vs Average CPU Time	40
Fig. 25 Optimum Tolerance with Respect to the Cost vs Time to Run	41
Fig. 26 Map-2 Solution Using RRT*	42
Fig. 27 Map-2 Solution Using the Proposed GB Gaussian RRT*	42
Fig. 28 Comparison Between Different Width Size to Average CPU Time	43
Fig. 29 Comparison Between Different Width Size to Average Cost	44
Fig. 30 Map-3 Solution Using the Proposed GB Gaussian RRT*	45
Fig. 31 Map-3 Solution Using the Proposed GB Gaussian RRT*	46
Fig. 32 Comparison Between Gap Size to Average CPU Time	47
Fig. 33 Comparison Between Different Gap Size to Average Cost	47
Fig. 34 Map-4 Solution Using RRT*	48
Fig. 35 Map-4 Solution Using the Proposed GB Gaussian RRT*	49
Fig. 36 Results of 100 Run for Trap Check Problem	50
Fig. 37 Map-5 Solution With RRT*	51
Fig. 38 Map-5 Solution with the Proposed GB Gaussian RRT*	52
Fig. 39 Results for Map 5 with 100 Independent Runs	52



## LIST OF ALGORITHMS

	Page
Algorithm 1: Pseudo Code for RRT Algorithm [21] .....	13
Algorithm 2: Pseudo Code for RRT* Algorithm [21] .....	24
Algorithm 3: Pseudo Code for the Proposed GB-Gaussian RRT* .....	31
Algorithm 4: Pseudo Code for GaussianSample (xstart, xend, cmax) .....	32

# CHAPTER 1

## INTRODUCTION

Path planning is one of the major aspects for fields such as Autonomous driving [1], graphical animation [2] and autonomous exploration [3]. Growth in autonomous vehicle research leads to efficient techniques in path planning. Computer efficiency increases rapidly which enables the use of autonomous vehicles in different areas, like warehouses, space exploration, transportation, and defense. Several path planning algorithms have been developed in recent times. The algorithm that can satisfy the system dynamics and obstacle avoidance and develop path from start point to goal point, if possible or return failure is considered as complete.

Path planning is a critical issue in the autonomous vehicle. There are different objectives of path planning, as listed below [4].

- Global planning
- Behavioral planning and
- Local planning

Global planning takes care of mission planning or route planning using different techniques such as D\* [4], A\* [4] and Probabilistic Road Map (PRM) [4] to find shortest and obstacle-free path. Behavioral planning focuses on event and maneuver management such as when to overtake. Local planning executes the decision taken by the behavioral planning in the manner that dynamic and kinematic model of the car taken consideration to make the ride more comfortable [4].

### 1.1 Path Planning and Trajectory Planning

Path planning and Trajectory planning are different but closely related problems of robotics. [5][6][7]. Path planning problem is to search a continuous curve (not required to be

smooth) in the configuration space, which starts at the initial point and ends at the goal point. The curve should follow the following criteria; (1) path not considering the time constraint, (2) create a path with the use of different segments and (3) each segment can be a trajectory. Trajectory planning is usually considered as the solution from the path planning algorithm and give a final decision for how to move on the path. Instead of path planning, trajectory planning considers the time constraint and determines the velocity and acceleration derivatives with respect to time. System kinodynamics also play a significant role to create an appropriate trajectory of the system.

## **1.2 Literature Review**

Researchers have focused on the path planning algorithms for many decades. Path planning problem is categorized in many ways based on the algorithm characteristics and their unique properties, such as Node based optimal algorithms, Sampling-based algorithms, Mathematics model-based algorithm, Bioinspired algorithm, etc. All these algorithms approach path planning problem in unique ways such that the sampling-based algorithms sample the configuration space using Monte Carlo sampling or using similar methods [8], a mathematical model Based algorithm uses all dynamics of the system, obstacle, and configuration space [9].

Node-based optimal algorithms are known as grid-based algorithms, that use uniform gridding in configuration space and explore through the decomposed graph [10]. As Fig. 1 shows, the most common node based algorithms are D\*, A\*, D\* Lite which are classified as discrete optimal planning [7], roadmap algorithm [11]. Dijkstra's Algorithm searches every node in the graph and store distance cost for that node from start point and create path [12]. D\* is an incremental search. D\* is an updated version of the Dijkstra's Algorithm. D\* is more efficient to get out of the local minima [12].

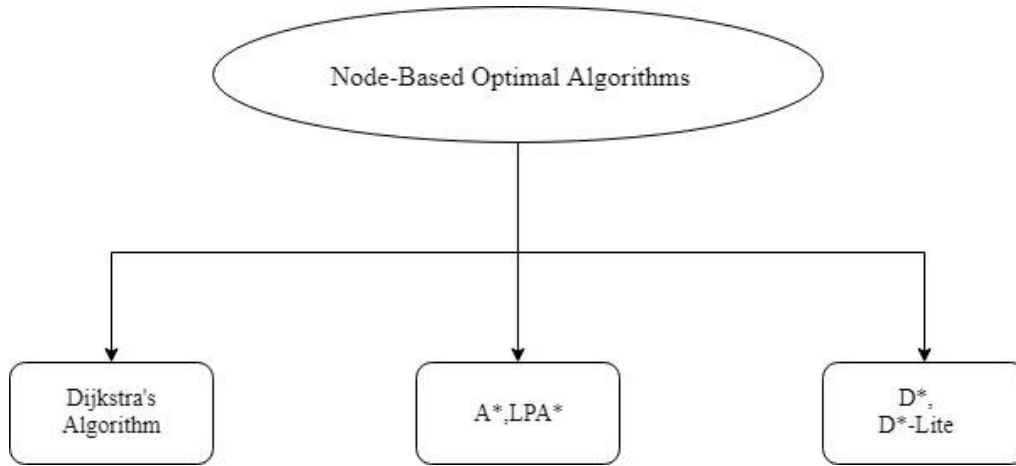


Fig. 1 Elements of Node Based Optimal Algorithms [13]

A\* is a node-based algorithm, which was implemented widely in different autonomous vehicles. i.e., hybrid A\* was implemented by a junior in DARPA challenge [14]. ]. A\* Algorithm is used in a static environment where the algorithm is continuously searching for the nearest node that approaches the destination [15]. A heuristic method is used to find the optimal path. The algorithm introduces three different variables which are H (Heuristic) value of the node, G value a movement cost of the node, and F value is an addition of H and G values. Open list and closed list of the nodes have been maintained all the time. A\* Algorithm has guaranteed solutions if it is possible in the graph. The disadvantage of the algorithm is the cost of the solution is very high in long and practical situations [16][17].

Lifelong Planning A\*(LPA\*) is an extended version of the A\*. LPA\* developed to work in real time applications [13]. D\*-Lite is a combination of D\* and LPA\*. D\*-Lite developed to work in real time applications, e.g, mobile robots [13].

Ant Colony Optimization (ACO) is the most popular bio-inspired algorithm. ACO is also one of the widely used techniques for small robots. ACO is a probabilistic search technique which can work in a static environment where the algorithm mocks the real ants [18]. Ants produce a pheromone that can be used for pheromone trail to indicate the optimal path. The system requires the previously known environment to perform [19].

Sampling-based algorithms, as listed in Fig. 2, are widely used recently for many autonomous vehicles. Rapidly Exploring Random Tree (RRT), Probabilistic Road Map (PRM) and Rapidly Exploring Random Graph (RRG) are sampling based algorithms as shown in Fig. 2. Sampling-based algorithms are categorized in two parts active and passive. The active algorithm returns the feasible path as an output. Passive Algorithms returns the graph as an output. Node-Based algorithms are used to find the feasible path out of many paths available in the graph [13]. PRM, RRG and K-Nearest PRM (K-PRM) are multiple query and passive algorithms. PRM, RRG and K-PRM returns the graph of many possible paths.

Artificial Potential Field Algorithm assumes the robot as particle reacts the potential field values. Endpoint assumes as an attractive field and obstacles as a repulsive field. Robot finds the path based on the attractive and repulsive fields [13].

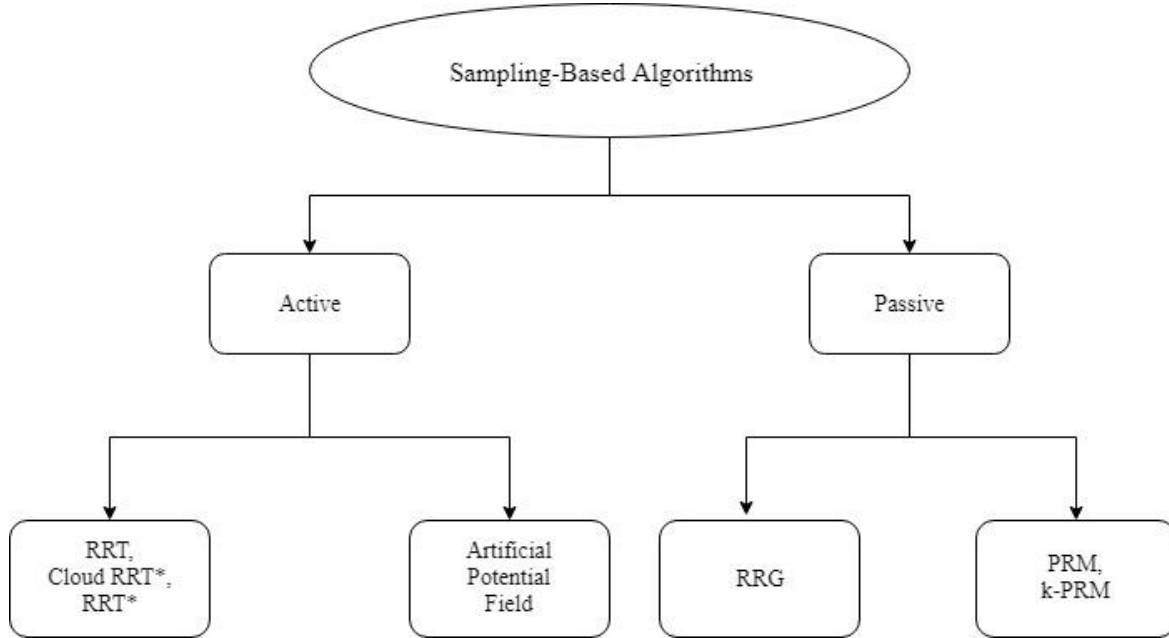


Fig. 2 Elements of Sampling-Based Algorithms [13]

RRT was proposed by, S. M. LaValle and J. J. Kuffner [20]. There is a graph of the randomly generated sampling points with the start and the goal point. The tree is developed with the use of samples to connect the start and the goal points. RRT algorithms do not need entirely modeled obstacles to avoid them and create an obstacle-free path. RRT\* is an improved version of the RRT which creates nodes alongside with the random sampling points [21][22]. New node works as a parent node and starts to find nearest node or sample point for tree expansion. This kind of heuristic method RRT\* can be used with the moving objects. The disadvantage of this sort of system is the tree expansion not focused to the goal, causing more time in execution [21].

Cloud RRT\* is a novel approach for biased sampling methods [23]. During this technique, the objective is to get more sampling density in promising regions for faster and more appropriate route planning. Cloud RRT\* is one of the ways to focus the tree expansion towards a goal instead

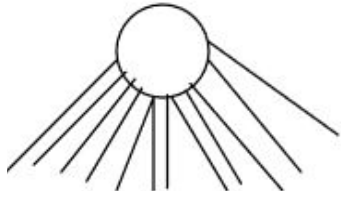
of expanding in the whole graph [23]. To achieve that case, a generalized Voronoi diagram is used to define objects in state-space. In the set of obstacle-free space, spherical cloud of the samples has been created.

Anytime RRTs, introduced by, i.e., D. Ferguson, A. Stentz, added a cost function in traditional RRT\* [24]. First path was developed with the use of RRT\*. Now, the cost function  $C_s$  was used to determine the distance between node to the start point. The algorithm only consider the nodes which have less  $C_s$  value and create a new path which is more cost efficient, but the computational price has to pay.

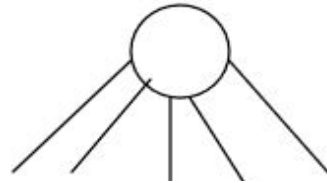
Informed RRT\* uses the ellipsoidal space to focus the search and the tree expansion [25]. Informed RRT\* is a simple but effective modification in RRT\* algorithm which helps to reduce the computational time. Tree expansion is more focused in informed RRT\* with compared to RRT\*, which results to lower CPU computational time.

### **1.3 Motivation**

As described in the last section, most path planning methods are non-heuristic. It is complicated to apply this methodology for the real-time and unknown environment. Sampling based-methodologies are designed to work in a specific environment. Non-sampling methods search every grid or continuous directions to avoid the obstacles which lead to time-consuming. As shown in Fig. 3 and Fig. 4, sampling-based methods use the probabilistic approach to avoid obstacles. Sampling-based methods are more efficient to spread in state space and easy to find path that follow the system dynamics. This characteristic motivated to research more in sampling-based methods. PRM and RRT are widely known and efficient sampling-based path planning methods. PRM is a multiple query search method and RRT is a single query search method. RRT is more efficient regarding memory, computational time and optimality, when compared to PRM [20][26].

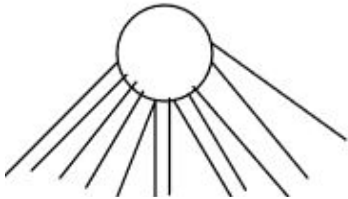


Probabilistically subsample edges

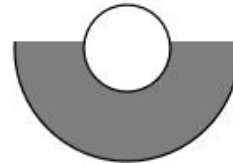


•A\* may try all edges in grid

Fig. 3 Grid Check Method for RRT And A\*



Probabilistically subsample edges



Continuum of choice

Fig. 4 Grid Check Method for RRT and Continous Method



## CHAPTER 2

### BACKGROUND

#### 2.1 Problem Formulation

The optimal planning problem is defined here similarly to [21]. Configuration space is defined as  $X = (0,1)^d$ ,  $d \in \mathbb{N}$ ,  $d \geq 2$ .  $\mathbb{N}$  is a set of natural numbers. For rest of the report  $d = 2$ , is in consideration. The start point for the problem  $x_{\text{start}}$  and the endpoint for the problem is  $x_{\text{end}}$ . Let  $X_{\text{obs}}$  is an obstacle region where  $X/X_{\text{obs}}$  an open set. Denote the obstacle-free region as  $X_{\text{free}} = \text{cl}(X/X_{\text{obs}})$ , where  $\text{cl}(\cdot)$  denotes the closure of a set.  $X_{\text{free}}$  is a close set of the open set  $X/X_{\text{obs}}$  so that all the points lie inside and on the border of the obstacle are excluded from  $X_{\text{free}}$ ,  $(x_{\text{start}}, x_{\text{end}}) \in X_{\text{free}}$ .  $(X_{\text{free}}, x_{\text{start}}, x_{\text{end}})$  triplet define the path planning problem.

The total variation of a path is reflecting its length. A function  $\sigma: [0,1] \rightarrow \mathbb{R}^d$  of bounded variation is considered as a path when it is continuous. The path is considered as collision-free when  $\sigma(\tau) \in X_{\text{free}}$ , for every  $\tau \in [0,1]$ . The collision-free path is feasible when  $\sigma(0) = x_{\text{start}}$ , and  $\sigma(1) = x_{\text{end}}$ . Feasibility problem is to find the feasible path which obeys the conditions in Eq (2.1) or reports failure. The total variation of the  $\text{TV}(\sigma)$  is defined below, where  $\sigma: [0,1] \rightarrow \mathbb{R}^d$ . The function  $\sigma$  provides bounded variation with  $\text{TV}(\sigma) < \infty$ .

$$\text{TV}(\sigma) = \sup_{n \in \mathbb{N}, 0=\tau_0 < \tau_1 < \dots < \tau_n=1} \sum_{i=1}^n |\sigma(\tau_i) - \sigma(\tau_{i-1})|. \quad (2.1)$$

Let  $\sigma: [0,1] \rightarrow X$  be a sequence of segments (a path) and  $\Sigma$  be the set of all non-zero paths. Optimal path planning problem is to find the path,  $\sigma^*$ , which searches paths between  $x_{\text{start}}$  and  $x_{\text{end}}$  and gives the minimum cost function,  $c: \Sigma \mapsto \mathbb{R} \geq 0$  and  $\sigma^* \in X_{\text{free}}$ ,  $\mathbb{R} \geq 0$  is the set of all non-negative real numbers.

$$\sigma^* = \arg \min_{\sigma \in \Sigma} \{c(\sigma) \mid \sigma(0) = x_{\text{start}}, \sigma(1) = x_{\text{end}}, \forall s \in [0,1], \sigma(s) \in X_{\text{free}} \}. \quad (2.2)$$

A function  $f(x)$  be the distance cost of the optimal path from  $x_{\text{start}}$  to  $x_{\text{end}}$  passing from  $x$ . Then the subset of segments that can improve the solution  $X_f \subseteq X$ . the current solution cost is  $c_{\text{best}}$ . The problem of search the most cost-effective path is equivalent to increasing the probability of adding a random state from  $X_f$  [20].

$$X_f = \{x \in X \mid f(x) < c_{\text{best}}\}. \quad (2.3)$$

## 2.2 Rapidly Exploring Random Tree (RRT) Algorithm

S. M. LaValle [20] introduced the Rapidly Exploring Random Tree (RRT) algorithm described in Algorithm 1 as a randomized data structure that is designed for path planning problems. This algorithm takes input a path planning problem  $(X_{\text{free}}, x_{\text{start}}, x_{\text{end}})$ ,  $n \in \mathbb{N}$ , and the cost function  $c : \Sigma \mapsto \mathbb{R} \geq 0$ . Planning Problem is given to the Algorithm 1 and the output returns in  $T = (V, E)$ , where  $V \in X_{\text{free}}$ , cardinality of set  $V$   $\text{card}(V) \leq n + 1$ , and  $E \in V \times V$ , where  $V$  represents the set of vertices and  $E$  represents the set of edges, the shortest-path algorithm can easily provide a solution for this graph. The relevant concepts are described.

**Sampling:** A function sample-free returns uniformly distributed random point  $x_{\text{rand}} \in X_{\text{free}}$  for every iteration in the algorithm. To generate the random sampling point, probability functions can be used. Nature of the probability function can manipulate the tree generation and the output of the algorithm. For this research, uniformly distributed random value generator from MATLAB toolbox was used.

$$\{\text{samplefree}_i(x_{\text{rand}})\}_{i \in \mathbb{N}} = \{\text{sample}_i(x_{\text{rand}})\}_{i \in \mathbb{N}} \cap X_{\text{free}} \quad (2.4)$$

**Distance:** As per the requirement of the system, RRT algorithms can work on the different distance metrics. Euclidean distance is taken into consideration during this research.

**Nearest Neighbor:** For the given graph  $T = (V, E)$ , where  $V \in X_{\text{free}}$ ,  $\text{card}(V) \leq n + 1$ , and  $E \in V \times V$ . Function  $x_{\text{nearest}} \leftarrow \text{Nearest}(T = (V, E), x_{\text{rand}})$  return the nearest point  $x_{\text{nearest}} \in V$ . To find the nearest point, various methods have been developed in data science. K-D Tree is one of the most efficient method to find the nearest point in the data set. Computational efficiency of the K-D tree is  $\log N$ , where  $N$  is the number of points in the data set [27]. Working of the K-D tree is very simple first, it is finding the median in the data set and split the data from that median. K-D tree algorithm repeats this process for all data point and creates a tree-like structure for the data set, shown as  $p$  is a data point,  $l$  is a median in Fig. 5, where the algorithm uses that tree to small regions in the data set. Now, to find nearest neighbor algorithm, only compare data in that region. Using this methodology, the algorithm reduces half points in every iteration. The nearest neighbor method is very critical for the success of RRT like algorithms.

$$\text{Nearest}(T = (V, E), x_{\text{rand}}) := \operatorname{argmin}_{v \in V} \|x - v\|. \quad (2.5)$$

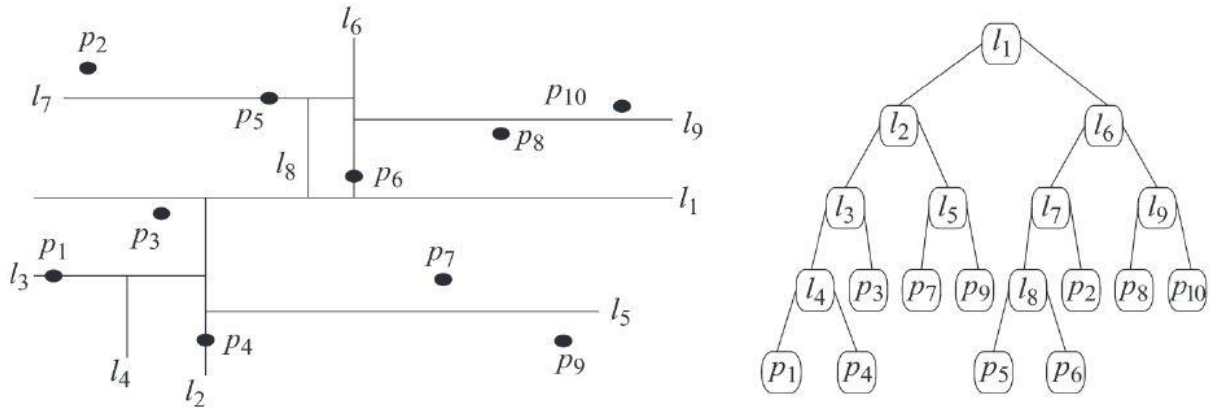


Fig. 5 Plane Subdivision and The Corresponding Tree Schematics [27]

**Steering:** Function  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}})$ , returns the  $x_{\text{new}} \in X_{\text{free}}$ , such that  $x_{\text{new}}$  is closer to  $x_{\text{nearest}}$  than  $x_{\text{rand}}$ . Prespecified step-size  $\eta \geq 0$ , is used to return  $\|x_{\text{new}} - x_{\text{nearest}}\| \leq \eta$ . Fig. 6 shows the working of the steer function.

$$\text{Steer}(x_{\text{nearest}}, x_{\text{rand}}) := \underset{x_{\text{new}} \in B_\eta}{\text{argmin}} \|x_{\text{new}} - x_{\text{nearest}}\| \quad (2.6)$$

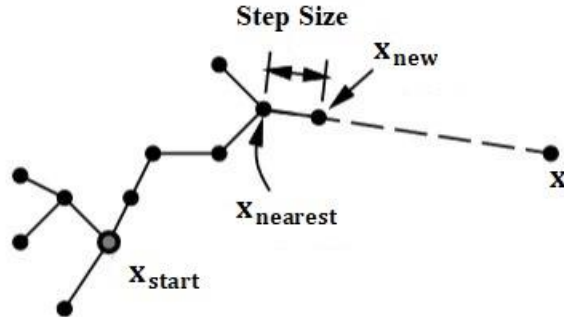


Fig. 6 Steering Function [20]

**Collision check:** Collision check is a Boolean function that returns true if any edge  $E \in X_{\text{free}}$  and false otherwise.

$$\text{obstacleFree}(x_{\text{nearest}}, x_{\text{new}}) = \begin{cases} 1, & (x_{\text{nearest}}, x_{\text{new}}) \in X_{\text{free}} \\ 0, & (x_{\text{nearest}}, x_{\text{new}}) \notin X_{\text{free}} \end{cases} \quad (2.7)$$

Algorithm 1 outlines the pseudocode for the RRT that can generate a path which can fulfill the algebraic constraints (taken from obstacles) and differential constraints (taken from nonholonomic and dynamics) of the system. Here, a points robot is taken in the considerations. RRT is a single-query algorithm. As shown in Fig. 7, the algorithm incrementally builds the tree of the possible trajectories, using steering function. Initially, the algorithm starts with the empty graph with  $x_{\text{start}}$  as first point in vertices set and zero points in edge set. As shown in Fig. 8, with every iteration sample-free function generates a random point  $x_{\text{rand}} \in X_{\text{free}}$ . Algorithm repeats the loop for  $n$  number of times, where  $n$  is a number of iterations in the Algorithm 1.

---

```

1:  $V \leftarrow \{x_{\text{start}}\};$ 
2:  $E \leftarrow \emptyset;$ 
3: for iteration=1.....n do
4:    $x_{\text{rand}} \leftarrow \text{SampleFree}_i;$ 
5:    $x_{\text{nearest}} \leftarrow \text{Nearest}(T = (V, E), x_{\text{rand}});$ 
6:    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}});$ 
7:   if  $\text{obstacleFree}(x_{\text{nearest}}, x_{\text{new}})$  then
8:      $V \leftarrow V \cup \{x_{\text{new}}\};$ 
9:      $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\};$ 
10:  End if
11: End for
12: return  $T = (V, E);$ 

```

---

Algorithm 1: Pseudo Code for RRT Algorithm [21]

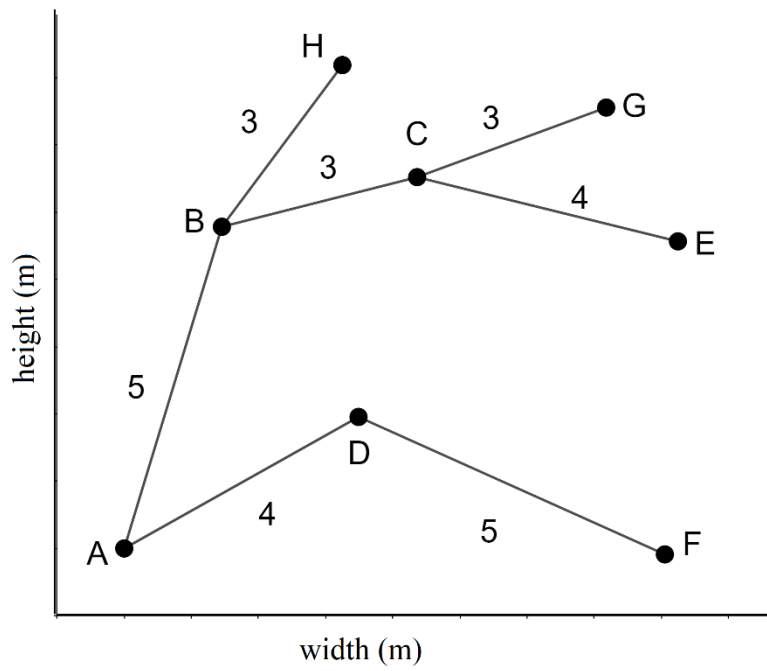


Fig. 7 Tree Expansion of RRT From the Start Point

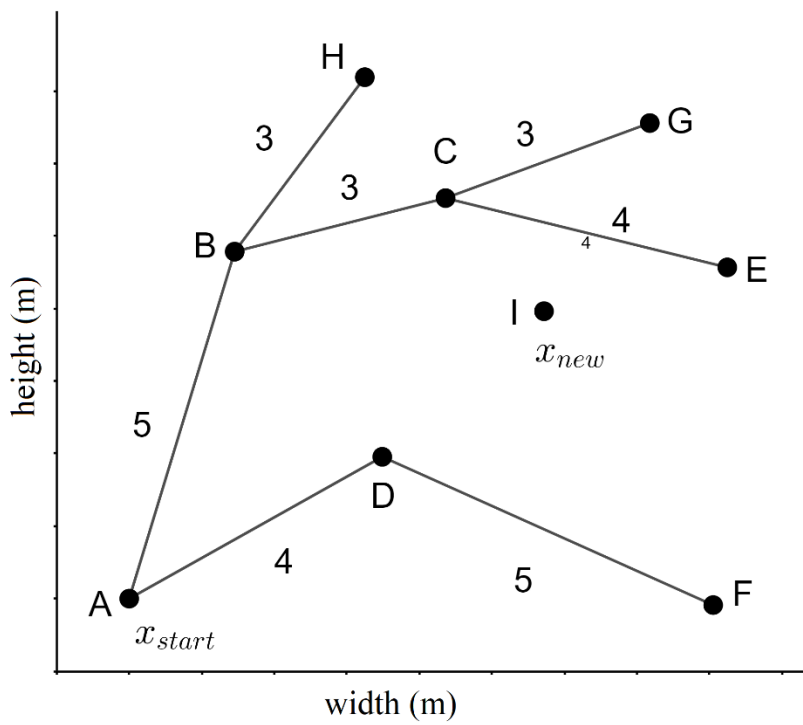


Fig. 8 New Point is Sampled

The nearest neighbor function finds the nearest point  $x_{nearest} \in V$  from the already available set of vertices as shown in Fig. 9. The algorithm tries to establish a connection between  $x_{nearest}$  and  $x_{rand}$  with the consideration that the trajectory between them is feasible. The steering function is used to establish this connection. Here, the predefined step-size  $\eta$  is used to generate a new point  $x_{new}$ . Step-size varies with the size of the state space so that algorithm can explore the state space more efficiently and within a smaller number of iterations. As shown in Fig. 10, using steering function and the proper step-size algorithm can explore different areas of the state space.

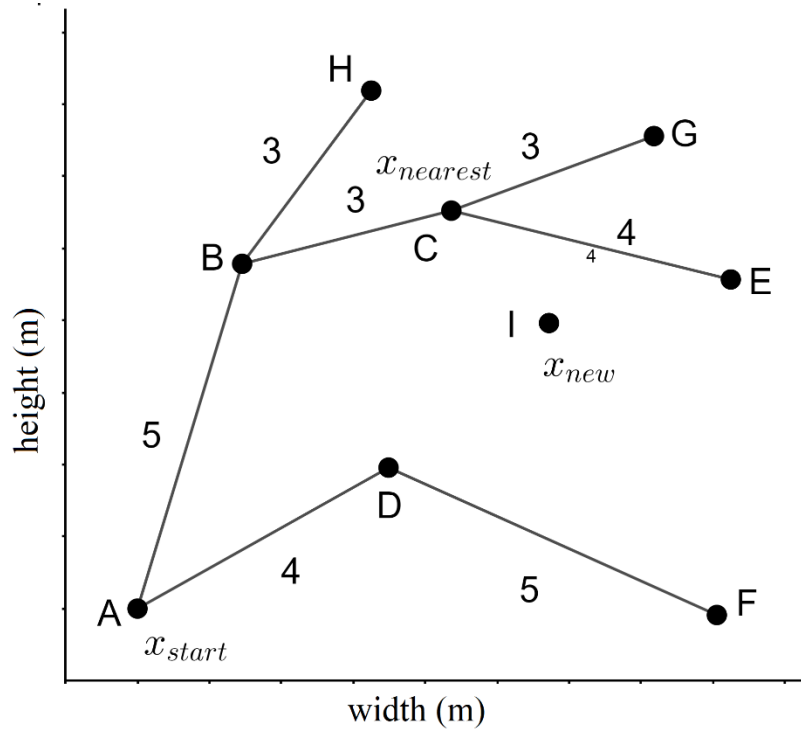


Fig. 9 Discovery of the Nearest Neighbor from the Vertex Set



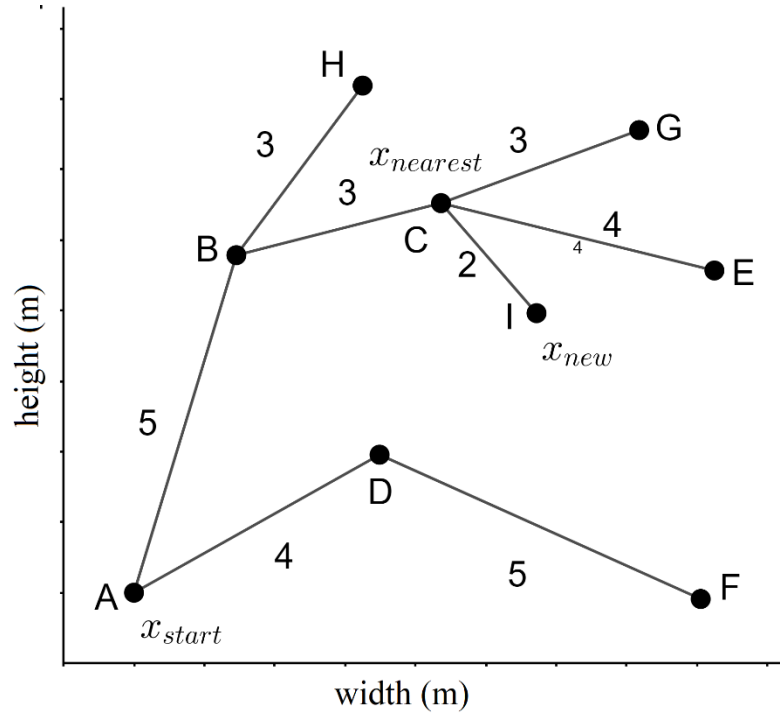


Fig. 10 Creation of a Connection Between Two Points

Fig. 11 shows the Voronoi diagram of the RRT vertices. As shown in Fig. 11, RRT is sensitive and biased to the large Voronoi regions before it covers the state space uniformly. The probability of the new vertex selection is directly proportional to the area of the Voronoi region.

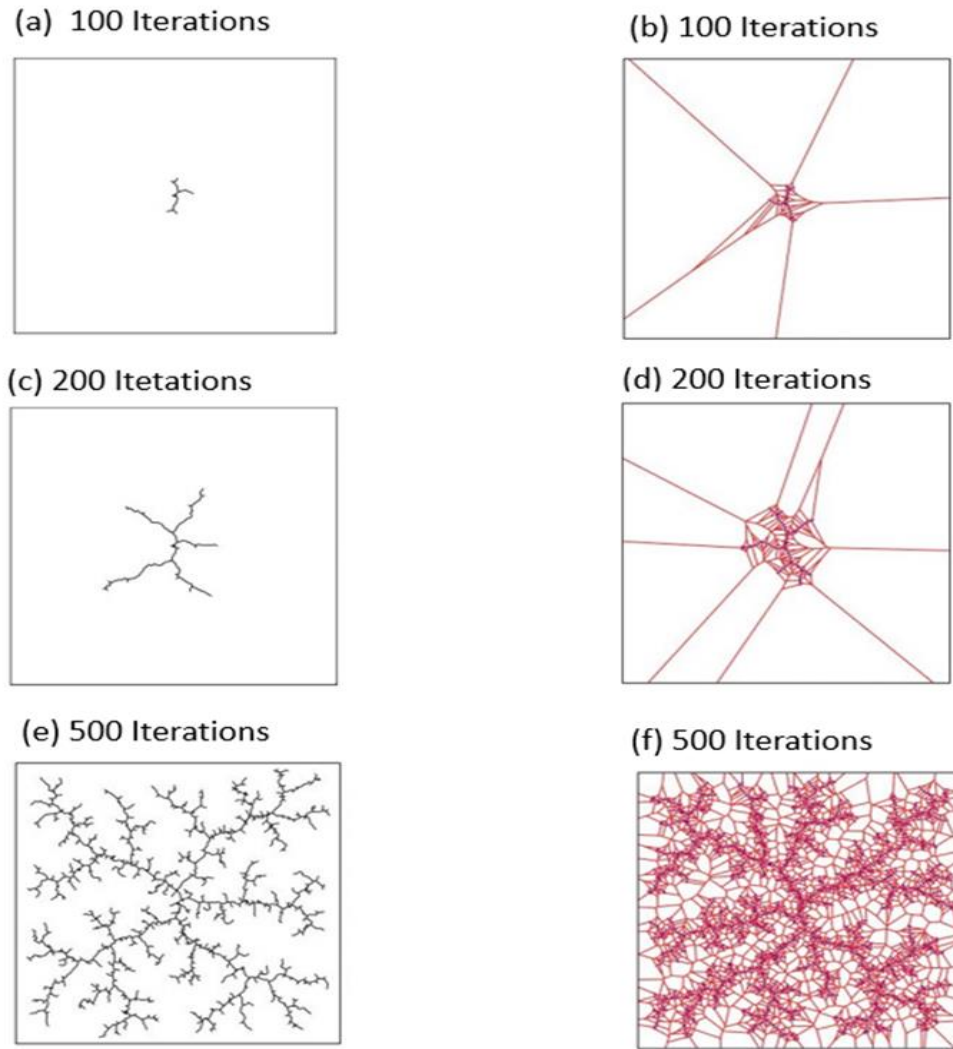


Fig. 11 The Relation Between RRT and Voronoi Region [28]

### 2.3 1<sup>st</sup> Layer of Optimization: Rewiring for Low-Cost Tree

Once the tree generated in the state space, the RRT planner only connects to the nearest neighbor that has no guarantee for cost efficiency. Using the first layer of optimization, a planner ensures that a newly generated point connects in such a way that the distance cost from the start point remain minimum. The relevant concepts are described.



As shown in Fig. 12, node labeled I is the newly generated point by the RRT planner. All the vertices lie in the circle generated from the radius  $r_{RRT^*}$  and center  $x_{new}$  using near vertices function trying to connect  $x_{new}$ .

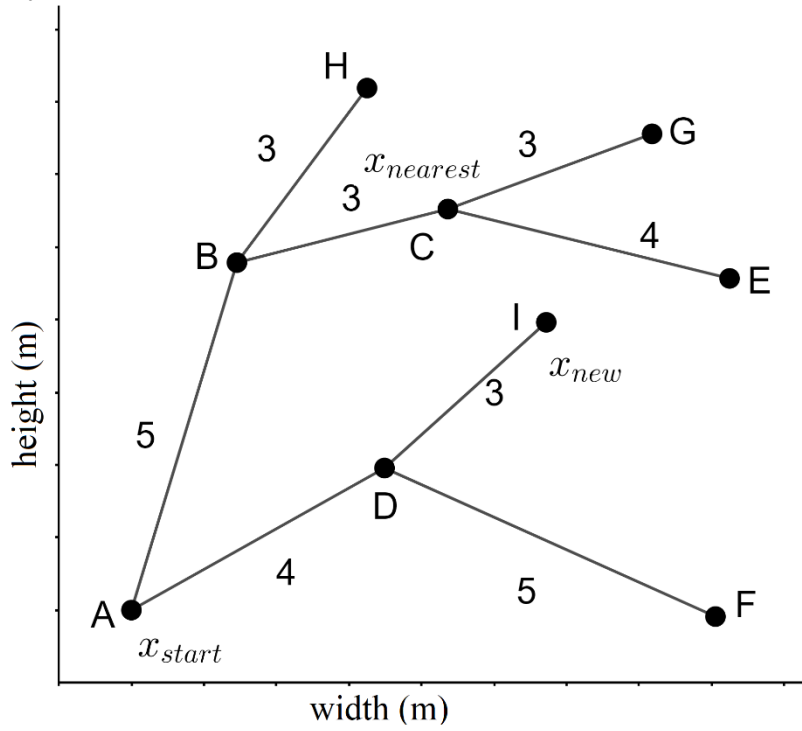


Fig. 13 Rewiring Process for New Tree

Before optimization, an RRT planner connects the start point to a new point through nearest point C as shown in Fig. 13. The path for that is  $A \rightarrow B \rightarrow C \rightarrow I$  and cost for that is 10. Now, after applying the first optimization, the planner tries to connect through D, E, G and cost for them are 7, 14.5, 14, respectively. The planner selects the minimum cost node to connect such that the distance cost from the start point is minimum. New path is  $A \rightarrow D \rightarrow I$  and the distance cost is 7.

## 2.4 2<sup>nd</sup> Layer of Optimization

2<sup>nd</sup> layer of optimization checks if  $x_{new}$  can reduce the cost of any already established tree by adding a new point in the existing tree. To perform this function near vertices with the same radius used in the 1<sup>st</sup> layer of optimization used to select the nodes from the set of trees. As shown in Fig. 14, a planner tries to establish a connection between  $x_{new}$  and other nodes in a circle, here I-C, I-E, I-G in the Fig. 14. Now, the algorithm compares the cost to reach the vertices from the start point through  $x_{new}$  and the cost of the already available trees to reach that node.

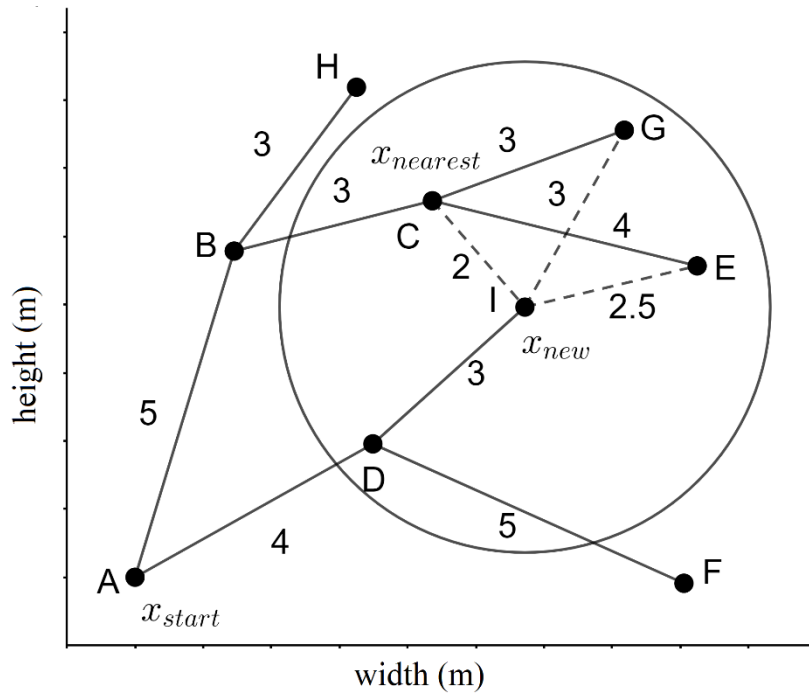


Fig. 14 Discover the Parent Nodes In the  $r$  Radius

As shown in Fig. 14, to reach vertices C, E, G cost from already available trees are respectively, 8, 12, 11, and the cost after applying optimization are respectively, 9, 9.5, 10. As shown in Fig. 14, distance cost to reach node E through  $x_{new}$  is reduced to 12 to 9.5, as shown in

Fig. 15, the algorithm connects the node E through node I ( $x_{new}$ ) and updates the tree to reach the node I.

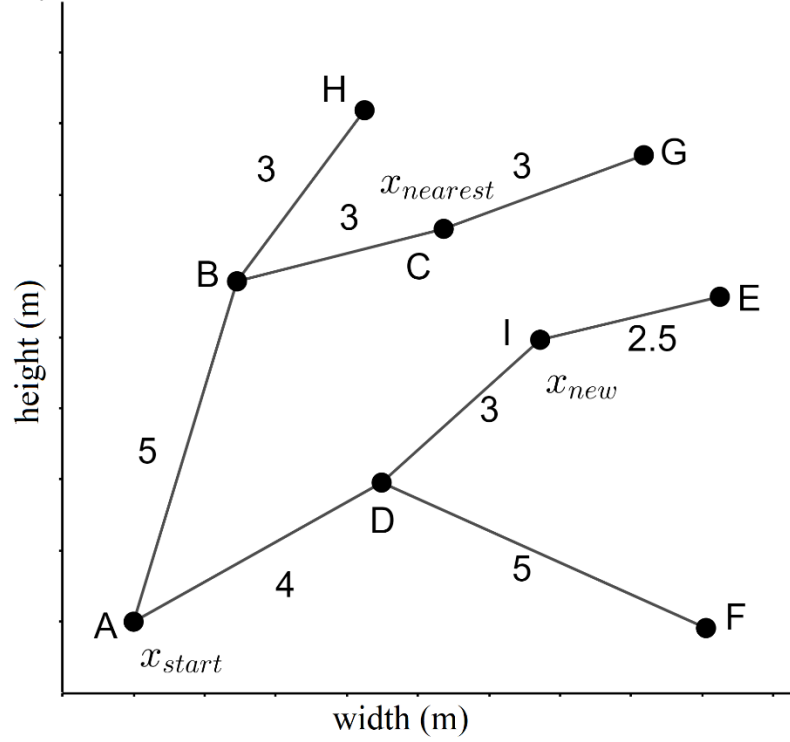


Fig. 15 Update the Parent Node for the Low-Cost Tree

## 2.5 Optimal RRT (RRT\*) Algorithm

RRT\* algorithm, given in Algorithm 2, is the asymptotically optimal, probabilistically complete, and computationally efficient version of the RRT and introduces some new functions. Near vertices work is the same as described. For any given two points  $x_1$  and  $x_2$ , the function  $\text{Line}(x_1, x_2)$  returns the straight-line path between two points.

The function  $\text{Parent}(u): V \rightarrow V$  returns the map for the vertex  $x \in V$  with a unique vertex  $u \in V$  so that  $(x, u) \in E$ .  $\text{Parent}(u)$  returns the set of nodes to reach the node  $u$  with the root node  $x$ . If the  $u = x_{start}$  in the  $\text{Parent}(u)$  then the  $\text{Parent}(x_{start}) = x_{start}$ . The function  $c.\text{Line}(x_1, x_2)$

returns the cost for moving from  $x_1$  to  $x_2$ . Here, the function provides the Euclidean distance between two points. The function  $\text{Cost}(x)$  returns the cost for the node  $x$  from the root node in the tree, for  $x = x_{\text{start}}$  cost for that  $\text{Cost}(x_{\text{start}}) = 0$ .

Initially same as RRT, it starts with the empty graph with  $x_{\text{start}}$  as the first vertex and the zero edges. With every iteration, the planner starts adding a vertex into the set of vertices and an edge into the set of edges. Same as RRT, using nearest and the steering function tree start growing in the state space. If the newly created edge between  $x_{\text{nearest}}$  and  $x_{\text{new}}$  is obstacle free, then the RRT\* algorithm apply the 1<sup>st</sup> and 2<sup>nd</sup> layer of optimization on the tree. First, it finds the vertices in the radius  $r_{\text{RRT}^*}$  using the near vertices function. The algorithm tries to connect the  $x_{\text{new}}$  with every vertex in the radius. For every obstacle free edge, it will compare the cost to reach the  $x_{\text{start}}$  through that vertex using the cost function. Once the algorithm finds the minimum cost tree, the planner updates the Parent Node, called the rewiring process.

Once the algorithm finds the minimum cost tree for  $x_{\text{new}}$ , it will apply the 2<sup>nd</sup> layer of optimization, for the same set of vertices. Again, the algorithm tries to connect  $x_{\text{new}}$  with every vertex in a circle. The algorithm checks for the function  $\text{ObstacleFree}$  for every possible edge. If the edge is obstacle free, it will compare the cost for every vertex to reach  $x_{\text{start}}$  through  $x_{\text{new}}$  and if any vertex reports the less cost, the algorithm updates edges for that vertex.

---

```

1:  $V \leftarrow \{x_{\text{start}}\}; E \leftarrow \emptyset;$ 
2:  $T = (V, E);$ 
3: for iteration=1.....n do
4:    $x_{\text{rand}} \leftarrow \text{SampleFree}_i;$ 
5:    $x_{\text{nearest}} \leftarrow \text{Nearest}(T = (V, E), x_{\text{rand}});$ 
6:    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}});$ 
7:   if  $\text{obstacleFree}(x_{\text{nearest}}, x_{\text{new}})$  then
8:      $V \leftarrow V \cup \{x_{\text{new}}\};$ 
9:      $X_{\text{near}} \leftarrow \text{Near}(T, x_{\text{new}}, r_{\text{RRT}*});$ 
10:     $x_{\text{min}} \leftarrow x_{\text{nearest}};$ 
11:     $c_{\text{min}} \leftarrow \text{Cost}(x_{\text{min}}) + c. \text{Line}(x_{\text{nearest}}, x_{\text{new}});$ 
12:    for  $\forall x_{\text{near}} \in X_{\text{near}}$  do
13:       $c_{\text{new}} \leftarrow \text{Cost}(x_{\text{near}}) + c. \text{Line}(x_{\text{near}}, x_{\text{new}});$ 
14:      if  $c_{\text{new}} < c_{\text{min}}$  then
15:        if  $\text{obstacleFree}(x_{\text{near}}, x_{\text{new}})$  then
16:           $x_{\text{min}} \leftarrow x_{\text{near}}; c_{\text{min}} \leftarrow c_{\text{new}};$ 
17:        End if
18:      End if
19:    End for
20:     $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\};$ 
21:    for  $\forall x_{\text{near}} \in X_{\text{near}}$  do
22:       $c_{\text{near}} \leftarrow \text{Cost}(x_{\text{near}}); c_{\text{new}} \leftarrow \text{Cost}(x_{\text{new}}) + c. \text{Line}(x_{\text{near}}, x_{\text{new}});$ 

```



```

23:         if  $c_{\text{new}} < c_{\text{near}}$  then
24:             if obstacleFree( $\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}}$ ) then
25:                  $\mathbf{x}_{\text{parent}} \leftarrow \text{Parent}(\mathbf{x}_{\text{near}})$ ;
26:                  $E \leftarrow E \setminus \{(\mathbf{x}_{\text{parent}}, \mathbf{x}_{\text{near}})\}; E \leftarrow E \cup \{(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}})\};$ 
27:             End if
28:         End if
29:     End for
30: End if
31: End for
32: return  $\mathbf{T} = (\mathbf{V}, \mathbf{E})$ ;

```

---

Algorithm 2: Pseudo Code for RRT\* Algorithm [21]

shows the tree expansion using the RRT\* algorithm that can reach to the different parts of the state space more efficiently and cost to reach to that point is less than RRT. There are some limitations with the RRT\* such as with the increase in the area of the state space time to find an optimal solution is increased because the RRT\* algorithm searches the entire state space which wastes time to search additional potentially irrelevant areas.

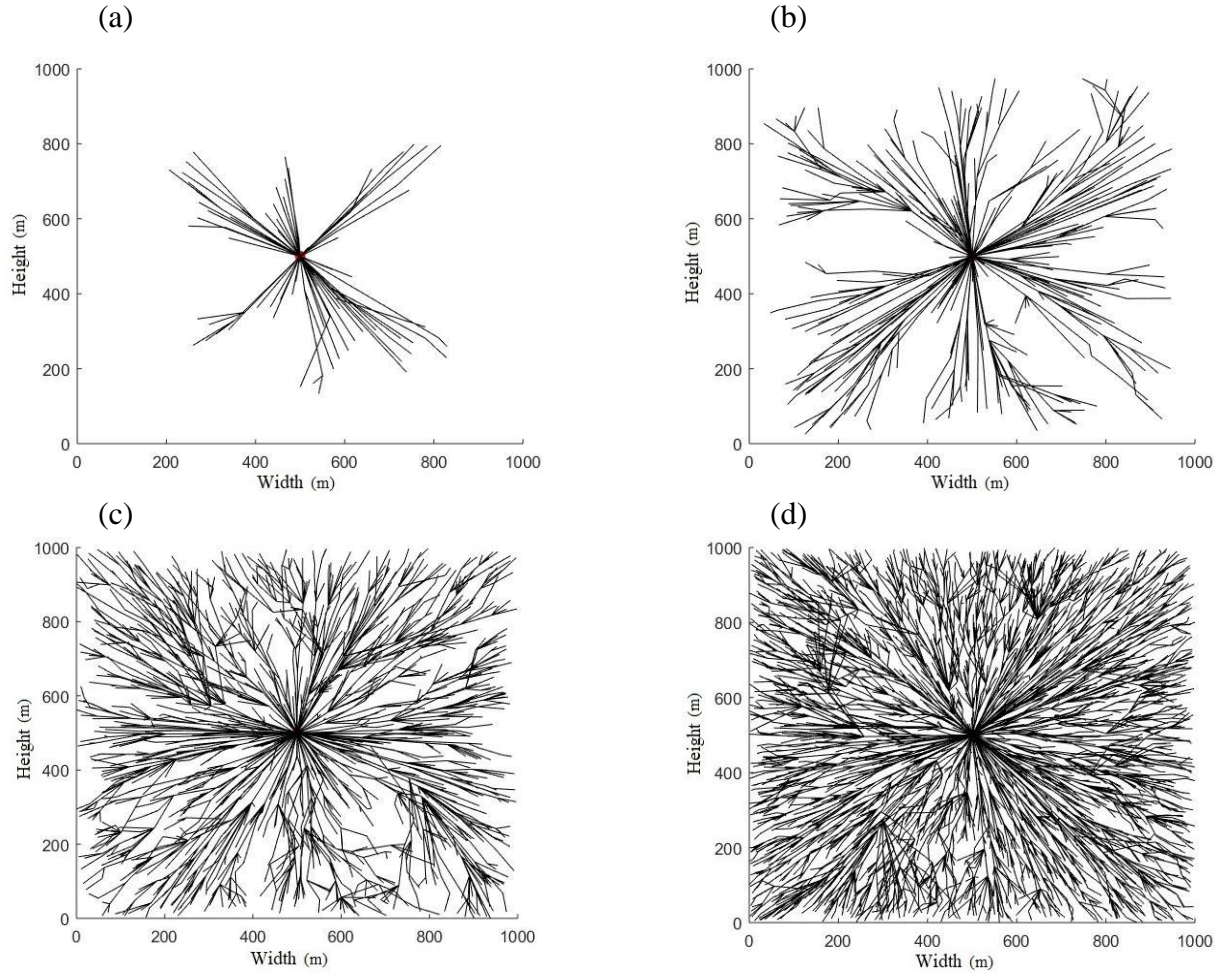


Fig. 16 RRT\* Tree Expansion (a) 100 Iterations, (b) 500 Iterations, (c) 2000 Iterations, (d) 5000 Iterations

## CHAPTER 3

### PROPOSED METHODOLOGY

Improving the efficiency of the RRT like algorithms, three important aspect of the algorithm must be improved; 1) Tree Expansion, 2) The direction of the Tree Expansion, 3) reduction of the relative state space. As discussed, the RRT\* algorithm mainly focuses on tree expansion. To improve the overall efficiency, the proposed algorithm addresses the other two problems that are not improved in RRT\*, lies 3<sup>rd</sup> optimization in the algorithm is Goal Biasing and 4<sup>th</sup> optimization in the algorithm is Gaussian Distribution.

#### 3.1 3<sup>rd</sup> Layer of Optimization: Goal Biasing

Goal Biasing optimization addresses the direction of the tree expansion problem. During the steering function, Eq (3.1) uses coordinates of  $x_{nearest}$  and  $x_{rand}$  to generate a  $x_{new}$  in the direction from  $x_{nearest}$  to  $x_{rand}$  with the predefined step-size  $\eta$ , formulated as,

$$\text{Steer}(x_{nearest}, x_{rand}, \eta) := \{x_{new} : x_{new} = x_{nearest} + (\eta \times \frac{(x_{rand} - x_{nearest})}{\|x_{rand} - x_{nearest}\|})\} \quad (3.1)$$

where  $\eta$  is a step-size,  $x_{nearest}$  is a coordinate of nearest neighbor,  $x_{rand}$  is a coordinate of the random point,  $x_{new}$  is the coordinate of the newly generated point.

Goal biasing approach depends on the two step-size  $q_1$  and  $q_2$  where  $q_1$  is the step size towards the random sampling point and the  $q_2$  is the step size towards the goal point. To avoid obstacles  $q_1$  and  $q_2$  has dynamic values. The function  $GB\text{Steer}(x_{nearest}, x_{rand}, q_1, q_2)$  returns the new point  $x_{new} \in X_{free}$ , which is biased to the goal region, written as

$$\begin{aligned}
& GBSteer(x_{nearest}, x_{rand}, q_1, q_2): x_{new} \\
& = x_{nearest} + \frac{q_1(x_{rand} - x_{nearest})}{\|x_{rand} - x_{nearest}\|} + \frac{q_2(x_{end} - x_{nearest})}{\|x_{end} - x_{nearest}\|} \quad (3.2)
\end{aligned}$$

where  $x_{nearest}$  is a coordinate of nearest neighbor,  $x_{rand}$  is a coordinate of the random point,  $x_{new}$  is the coordinate of the newly generated point.  $x_{end}$  is the coordinate of end point.

When  $x_{rand} \in X_{free}$ , let  $q_2 > q_1$  lead the tree expansion towards the goal point and rate of goal searching speed up. If the planner finds the obstacle in few iterations, it swaps the values of  $q_1$  and  $q_2$ .  $q_1 > q_2$  guides the search in the random point direction. This dynamic behavior of  $q_1$  and  $q_2$  helps the system to eventually get out of the local minima.

### 3.2 4<sup>th</sup> Layer of Optimization: Gaussian Distribution

Gaussian Distribution is a solution to reduce the relative state space. Using this methodology, tree expansion can be bounded with some area to speed the path planning problem. Gaussian Distribution widely used in data science for clustering of data [29][30][31]. Eq 3.3 shows the mathematical representation of the multivariant Gaussian distribution. Eq 3.4 and Eq 3.5 represent the expansion of the Gaussian distribution regarding the eigenvectors. Fig. 17 shows the Multivariant Gaussian Distribution, the shape of the Gaussian distribution is depending on the  $c_{best}$  and  $c_{min}$ .

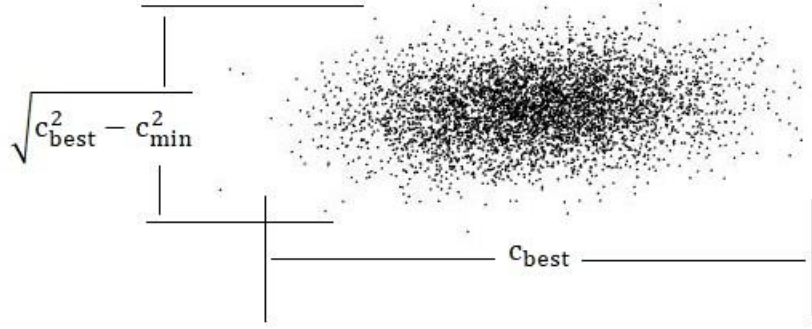


Fig. 17 Gaussian Distribution Sampling [32]

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{2\pi^{d/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right\} \quad (3.3)$$

$$\Sigma = U \Lambda U^T \quad (3.4)$$

$$\Sigma = \begin{bmatrix} u_1 & u_2 \\ \downarrow & \downarrow \end{bmatrix} \begin{bmatrix} \lambda_1 & \\ & \lambda_2 \end{bmatrix} \begin{bmatrix} u_1 & \rightarrow \\ u_2 & \rightarrow \end{bmatrix} \quad (3.5)$$

where  $\mu$  is a  $d$ -dimensional mean vector,  $\Sigma$  is a  $D \times D$  covariance matrix,  $U$  is a matrix whose row are  $u_1$  and  $u_2$  and  $U^T$  is an inverse matrix of  $U$  such that  $UU^T = I$ ,  $I$  is the identity matrix,  $|\Sigma|$  is a determinant of  $\Sigma$ ,  $u_1, u_2$  are the major axis eigenvectors of the covariance matrix with corresponding eigenvalues  $\lambda_1, \lambda_2$  [31][32].

### 3.3 Proposed Goal Biased – Gaussian RRT\* Algorithm

The proposed GB-Gaussian RRT\* algorithm addresses all three sampling-based path planning problems. First, some new functions are introduced for this proposed algorithm. The function `GaussianSample(xstart, xend, cbest)` returns the random sampling point with the Gaussian

sampling. Gaussian sampling is the key methodology for this algorithm. The function GoalRegion is a binary function which reports true if the  $x_{\text{new}}$  lies within the goal region. Goal region is the predefined value. The function SampleUnitBall returns the sampling points unit ball radius. These sample points are saved in variable  $x_{\text{ball}}$  that used in Gaussian sampling for the reference points. Using the Eq 3.4 and Eq 3.5, the function  $\Sigma(x_{\text{ball}}, (\lambda_1, \lambda_2), x_{\text{center}})$  returns the sampling point that follows the gaussian distribution properties.

Initially, this algorithm works as the same as RRT\* with  $x_{\text{start}}$  as vertex and empty edges. With every iteration, tree starts expands, but the algorithm has a unique property that helps the planner to generate the  $x_{\text{new}}$  a point not only in the direction of the random point but it is biased to the goal region. This biased nature of the algorithm helps to reach the first solution more quickly. Once the first solution is available, the algorithm uses that path as a reference to generate sample points to find a path.  $X_{\text{sol}} \in T$ ,  $T$  is a set of all available solutions. As shown in the Fig. 17, the  $c_{\text{best}}$  and  $c_{\text{min}}$  values are used to perform function GaussianSample where  $c_{\text{best}}$  is a minimum cost solution from all available solutions and  $c_{\text{min}}$  is the Euclidean distance between  $x_{\text{start}}$  and  $x_{\text{end}}$  points. As shown in Algorithm 4 line 6, first SampleUnitBall generates the sample point. Now,  $\lambda_1 = c_{\text{best}}/2$  and  $\lambda_2 = \left( \sqrt{c_{\text{best}}^2 - c_{\text{min}}^2} \right)$  while  $u_1, u_2$  are the rotation matrix taken from the imaginary line between start and end point. The value of  $\mu = x_{\text{center}}$  is the center of this sampling.  $\lambda_1, \lambda_2, x_{\text{ball}}$  and  $x_{\text{center}}$  values are given as inputs for the function  $\Sigma(x_{\text{ball}}, (\lambda_1, \lambda_2), x_{\text{center}})$  that returns the random point. Algorithm 4 was developed during the research to generate the Gaussian distribution using the available path data.

The number of iterations is increasing the cost for the  $c_{\text{best}}$  that decreases eventually as the result value of the  $\lambda_2$  also decreases. If there are no obstacles in the system theoretically, after  $n$

number of iterations, the value of  $\lambda_2$  falls near to zero and the algorithm provides the straight line between start and end point, i.e.,  $c_{best} \cong c_{min}$ . Algorithm 3 shows the pseudo code for the proposed GB Gaussian RRT\*. Algorithm 3 was modified from the Algorithm 2 during the research so that it can perform the proposed GB Gaussian RRT\* methodology.

---

```

1:  $V \leftarrow x_{start}$ ;
2:  $E \leftarrow \emptyset$ ;
3:  $T = (V, E)$ ;
4:  $X_{sol} \leftarrow \emptyset$ ;
5: for iteration=1.....n do
6:    $x_{rand} \leftarrow \text{GaussianSample}(x_{start}, x_{end}, c_{best})$ ;
7:    $x_{nearest} \leftarrow \text{Nearest}(T = (V, E), x_{rand})$ ;
8:    $x_{new} \leftarrow \text{GBSteer}(x_{nearest}, x_{rand}, q_1, q_2)$ ;
9:   if  $\text{obstacleFree}(x_{nearest}, x_{new})$  then
10:     $V \leftarrow V \cup \{x_{new}\}$ ;
11:     $X_{near} \leftarrow \text{Near}(T, x_{new}, r_{RRT*})$ ;
12:     $x_{min} \leftarrow x_{nearest}$ ;
13:     $c_{min} \leftarrow \text{Cost}(x_{min}) + c \cdot \text{Line}(x_{nearest}, x_{new})$ ;
14:    for  $\forall x_{near} \in X_{near}$  do
15:       $c_{new} \leftarrow \text{Cost}(x_{near}) + c \cdot \text{Line}(x_{near}, x_{new})$ ;
16:      if  $c_{new} < c_{min}$  then
17:        if  $\text{obstacleFree}(x_{near}, x_{new})$  then
18:           $x_{min} \leftarrow x_{near}$ ;

```

```

19:           $c_{\min} \leftarrow c_{\text{new}};$ 
20:      End if
21:  End if
22: End for
23:  $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\};$ 
24: for  $\forall x_{\text{near}} \in X_{\text{near}}$  do
25:      $c_{\text{near}} \leftarrow \text{Cost}(x_{\text{near}});$ 
26:      $c_{\text{new}} \leftarrow \text{Cost}(x_{\text{new}}) + c. \text{Line}(x_{\text{near}}, x_{\text{new}});$ 
27:     if  $c_{\text{new}} < c_{\text{near}}$  then
28:         if obstacleFree( $x_{\text{new}}, x_{\text{near}}$ ) then
29:              $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}});$ 
30:              $E \leftarrow E \setminus \{(x_{\text{parent}}, x_{\text{near}})\};$ 
31:              $E \leftarrow E \cup \{(x_{\text{new}}, x_{\text{near}})\};$ 
32:         End if
33:     End if
34: End for
35: if goalregion ( $x_{\text{new}}$ ) then
36:      $X_{\text{sol}} \leftarrow X_{\text{sol}} \cup \{x_{\text{new}}\};$ 
37:      $c_{\text{best}} \leftarrow \min x_{\text{sol}} \in x_{\text{sol}} \{\text{Cost}(x_{\text{sol}})\};$ 
38: End if
39: End for
40: return  $T = (V, E);$ 

```

---

Algorithm 3: Pseudo Code for the Proposed GB-Gaussian RRT\*



---

```

1: if  $c_{max} > 0$  then
2:    $c_{min} = \|x_{end} - x_{start}\|_2$ ;
3:    $x_{center} \leftarrow (x_{start} + x_{end})/2$ ;
4:    $\lambda_1 \leftarrow c_{max}/2$ ;
5:    $\lambda_2 \leftarrow \left( \sqrt{c_{best}^2 - c_{min}^2} \right) / 2$ ;
6:    $x_{ball} \leftarrow SampleUnitBall$ ;
7:    $x_{rand} \leftarrow \Sigma(x_{ball}, (\lambda_1, \lambda_2), x_{center})$ ;
8: else
9:    $x_{rand} \leftarrow SampleFree_i$ 
10: end if
11: return  $x_{rand}$ ;

```

---

Algorithm 4: Pseudo Code for GaussianSample ( $x_{start}, x_{end}, c_{max}$ )

## CHAPTER 4

### RESULTS

This chapter shows the simulation results for the RRT\* and proposed GB Gaussian RRT\* algorithms using the MATLAB software. All the simulation experiments were performed in Windows 10 on an Intel i5-7200U CPU @ 2.5GHz with 8GB RAM, to show the algorithm operations and how they behave in different situations. Here, five different maps were taken into considerations. For all the experiments the number of iterations  $n = 1500$  and the step size  $\eta = 3\%$  of the width of the map. And for the proposed GB Gaussian RRT\*,  $q_1 = 3\%$  and  $q_2 = 5\%$  of the width of the map were taken into considerations. Map size for all experiments is  $1000 \times 1000 \text{ m}^2$  except different size for no obstacle test. Cost is representing the distance cost value and time is represents the computational time. All the distance cost values during research are in meters. 100 Independent runs were performed to gather output data for every experiment. Procedure to generate pseudo random point seed and map remain same for RRT\* and the proposed GB Gaussian RRT\* algorithms during experiments.

Fig. 18 and Fig. 19 show implementation of the RRT\* and the proposed GB Gaussian RRT\*, respectively. Initial parameters for both algorithms were the same. The RRT\* algorithm required 463 iterations to find first path while the proposed GB Gaussian RRT\* algorithm uses goal biasing approach and find first path in just 164 iterations. Path data is used as an input, to generate a Gaussian distribution for the proposed GB Gaussian RRT\* and limit the state space for tree expansion in the proposed GB Gaussian RRT\*, as shown in Fig. 19. Instead RRT\* still uses full state space for tree expansion, as shown in Fig. 18.

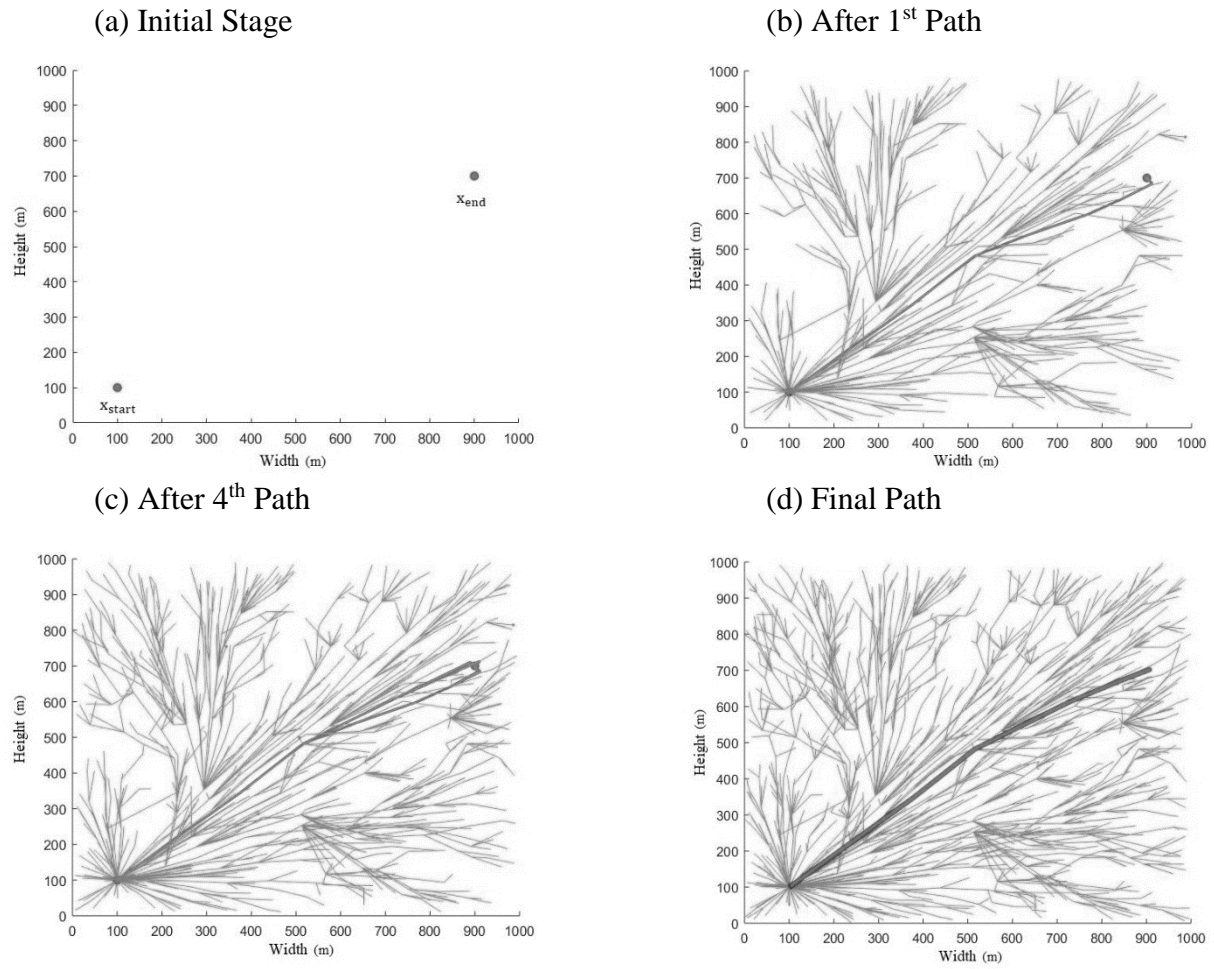


Fig. 18 RRT\* Implementation at Different Stages

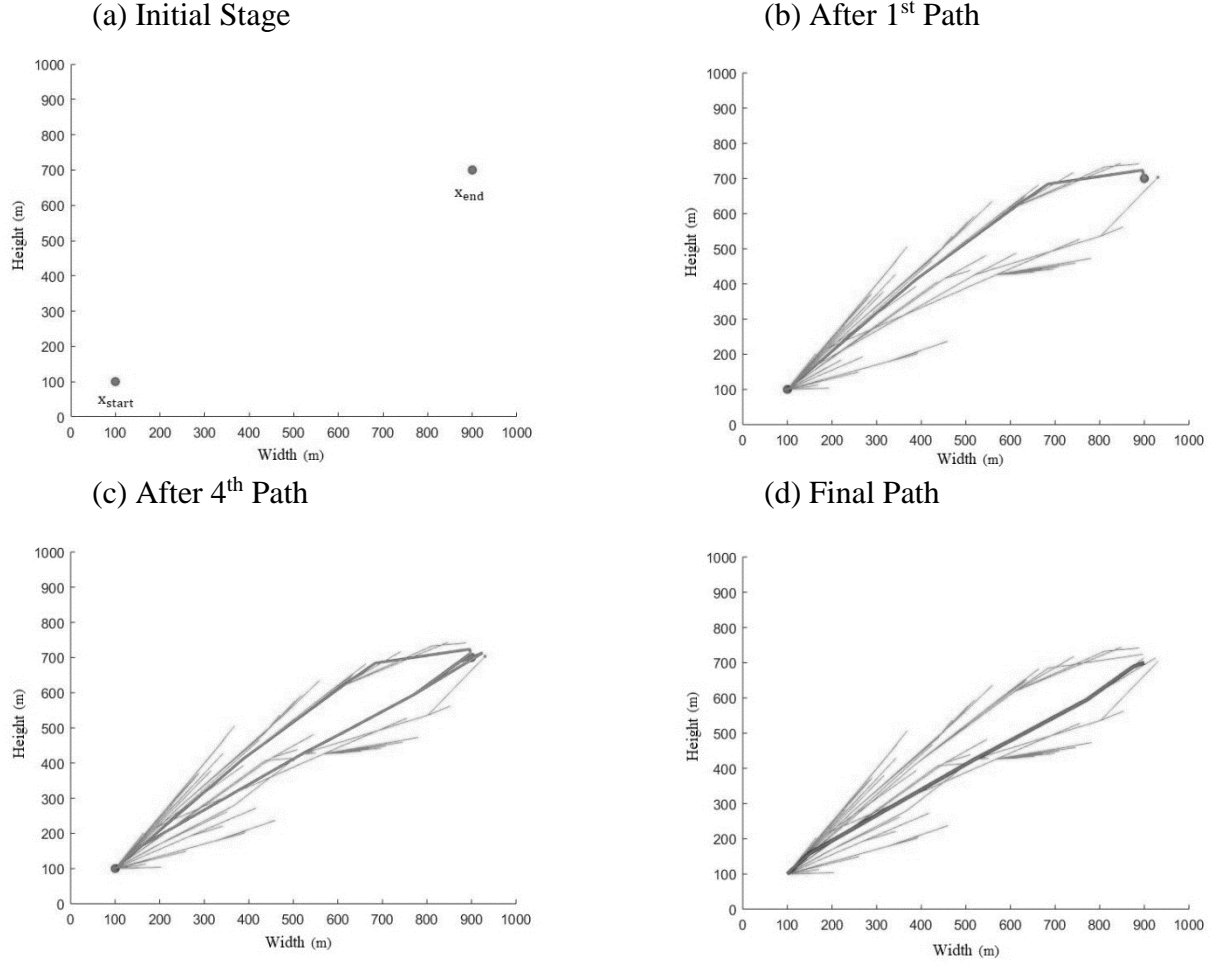
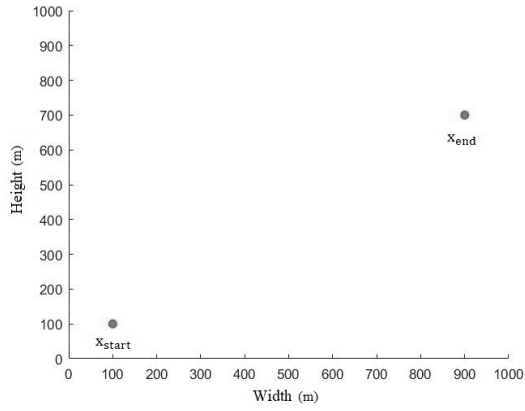


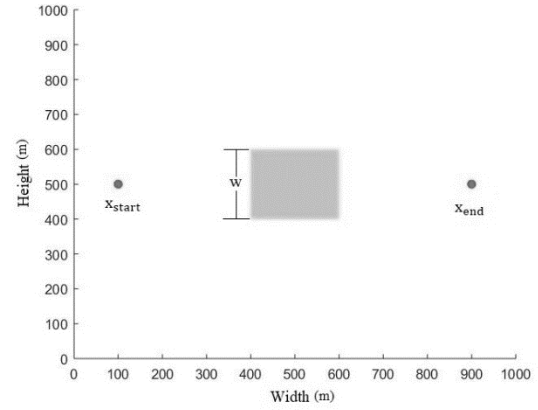
Fig. 19 The Proposed GB Gaussian RRT\* Implementation at Different Stages

Fig. 20 shows the different map configurations. Map-1 is a no Obstacle map. Changing state space shows the abilities of both algorithms to find the best path in less time. Map-2 is a width effect map and changing the width of the obstacle both algorithms show its capabilities. Map-3 is a gap effect map where gap size changes. Map-4 is a two-obstacle map to check traps. Map-5 is T type obstacle map.

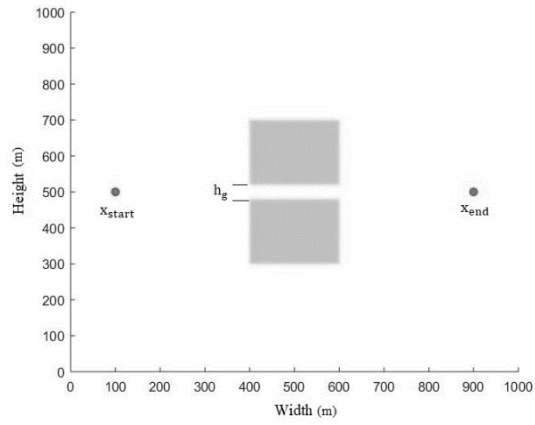
(a) Map 1 – No Obstacle



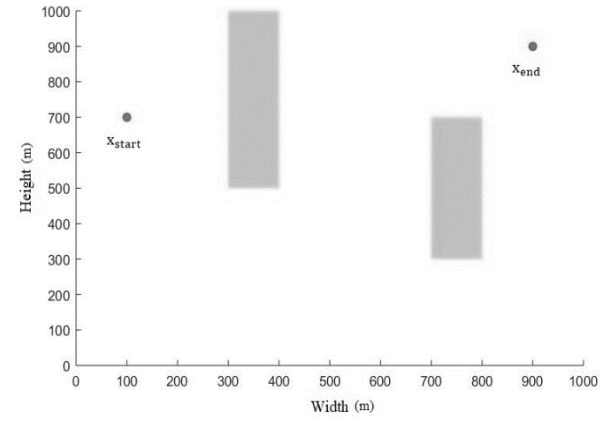
(b) Map 2 – Width Effect



(c) Map 3 – Narrow Gap



(d) Map 4 – Two Obstacle for Trap



(e) Map 5 – T Type obstacle

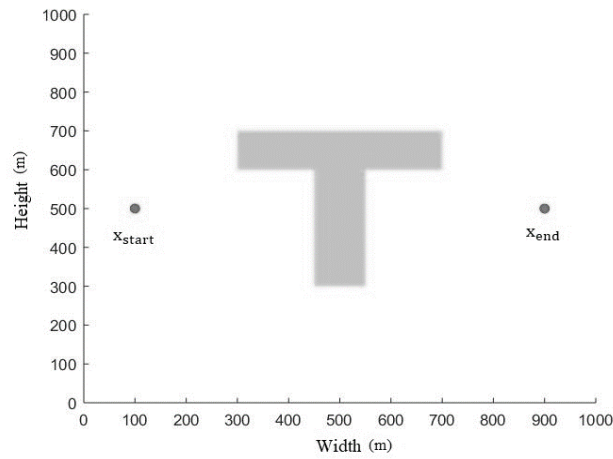


Fig. 20 (a) Map-1 (b) Map-2 (c) Map-3 (d) Map 4 (e) Map 5

#### 4.1 Map-1: No Obstacle

Fig. 21 shows the implementation of the RRT\* algorithm with the starting point  $x_{\text{start}} = (100,100)$  and the end point  $x_{\text{end}} = (900,700)$  at no obstacle environment Map-1. Size of the map is  $1000 \times 1000 \text{ m}^2$  and the distance between the start to the end point is 1000. To find the path between the start to the endpoint, RRT\* algorithm took 7.50 sec and cost for the path is 1119.85. As shown in Fig. 21, the algorithm searches the entire state space and need more time to find the near optimal path. Instead of RRT\* algorithm, the proposed GB Gaussian RRT\* took much less time and found the more cost-effective path. As shown in Fig. 22, the proposed GB Gaussian RRT needs 0.215 sec and the cost for that path is 1000.69. The proposed GB Gaussian RRT\* focused to the endpoint and saved initial time to search path. Once the initial path is developed, the planner limits the search area near to the first path and saves time and provides a more cost-effective path.

As shown in Fig. 23 the path in black is the ideal path and the path in green is developed using RRT\* and the path in red is developed using the proposed GB Gaussian RRT\*. The path developed using the proposed GB Gaussian RRT\* is near optimum.

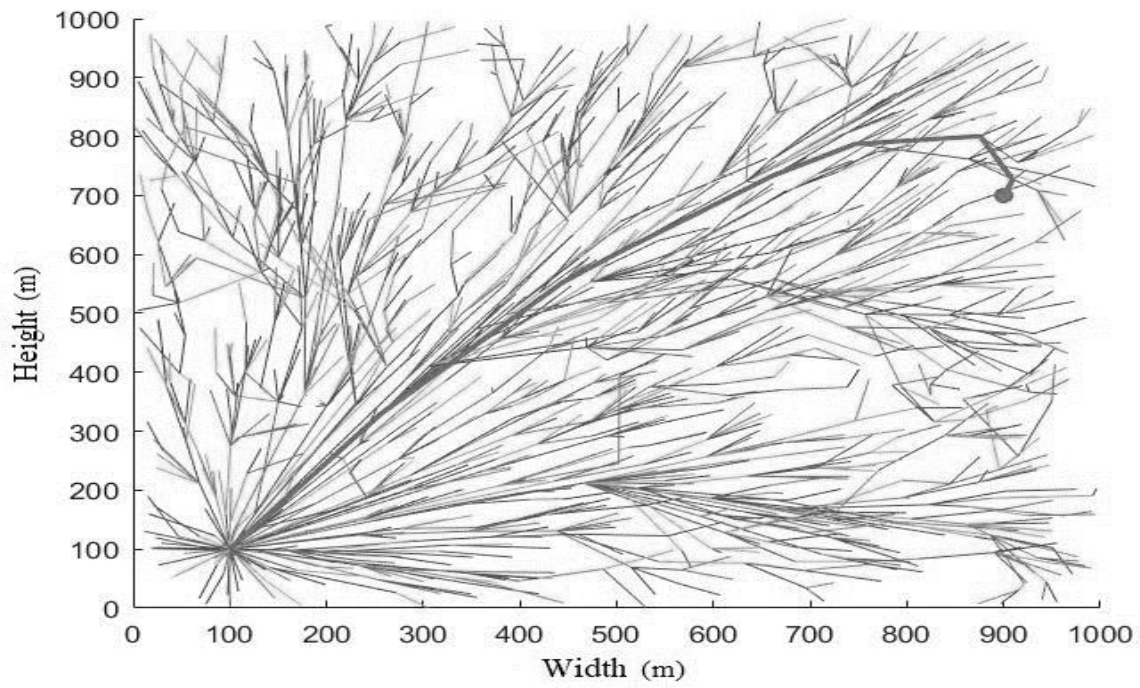


Fig. 21 Map-1 Solution Using RRT\*

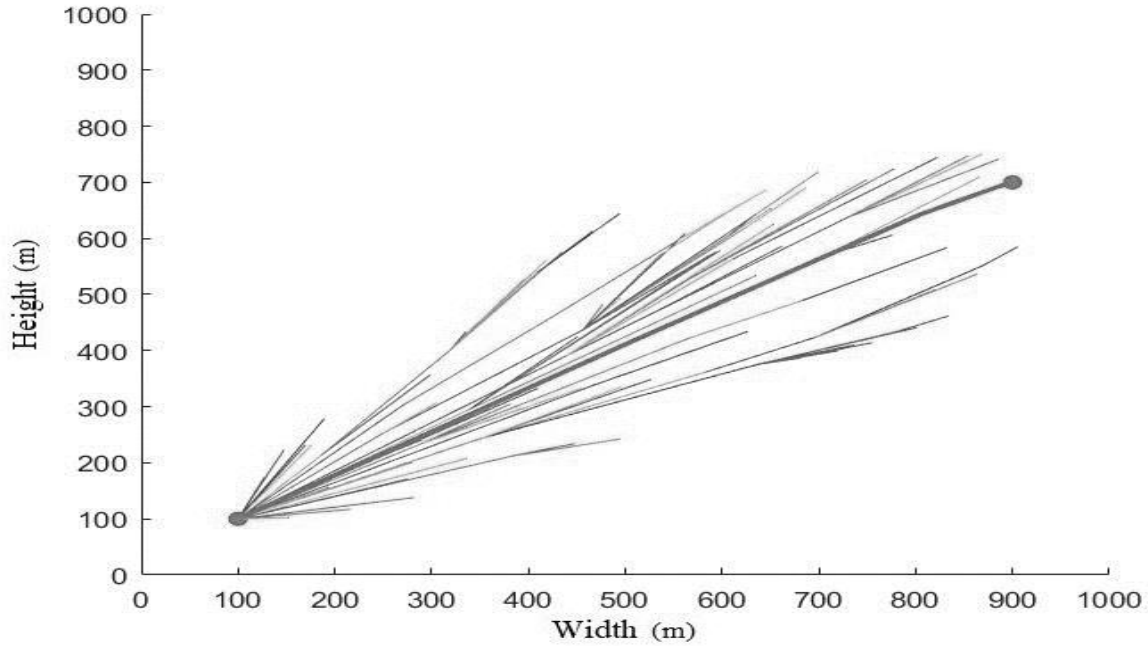


Fig. 22 Map-1 Solution Using the Proposed GB Gaussian RRT\*

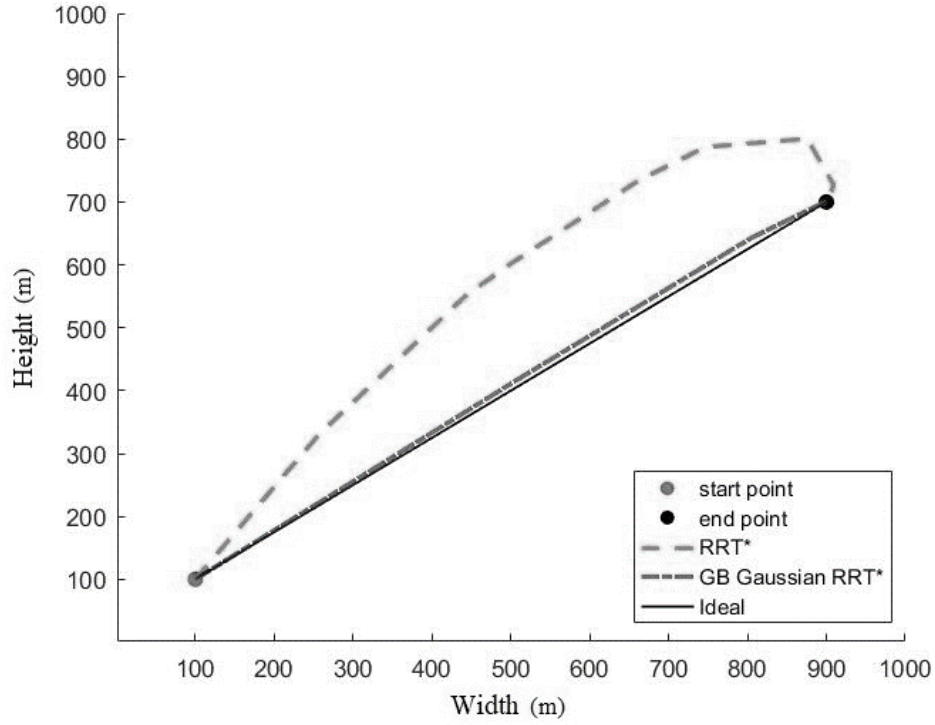


Fig. 23 Comparison Between GB Gaussian RRT\*, RRT\* and Ideal Path

Fig. 24 shows the comparison between RRT\* and the proposed GB Gaussian RRT\* with no obstacle maps. Here  $h/d$  represents the ratio between the height of the state space and the distance to the endpoint from the start point.  $d = 1000$  m and the height of the state space changes with the experiments. For no obstacle, 1000 m are the ideal cost for the path. For every  $h/d$  values, both algorithms performed 100 times. For  $h/d = 1$ , RRT\* algorithm needs average 4.681 sec and the average cost 1025.36 m and the proposed GB Gaussian RRT\* need average 0.364 sec and the average cost 1008.8 m. Same for ratio 2, 3, 4, RRT\* needs average 5.832 sec, 6.11sec, 7.065 sec, respectively, and average cost is 1048.91m, 1057.57m, 1065.21m and the proposed GB Gaussian RRT\* need average 0.176sec, 0.153sec, 0.150sec and average costs are 1008.73m, 1008.02m,



1008.85m, respectively. The experiments show that the proposed GB Gaussian RRT\* is independent of the size of the state space where RRT\* is dependent on the state space size.

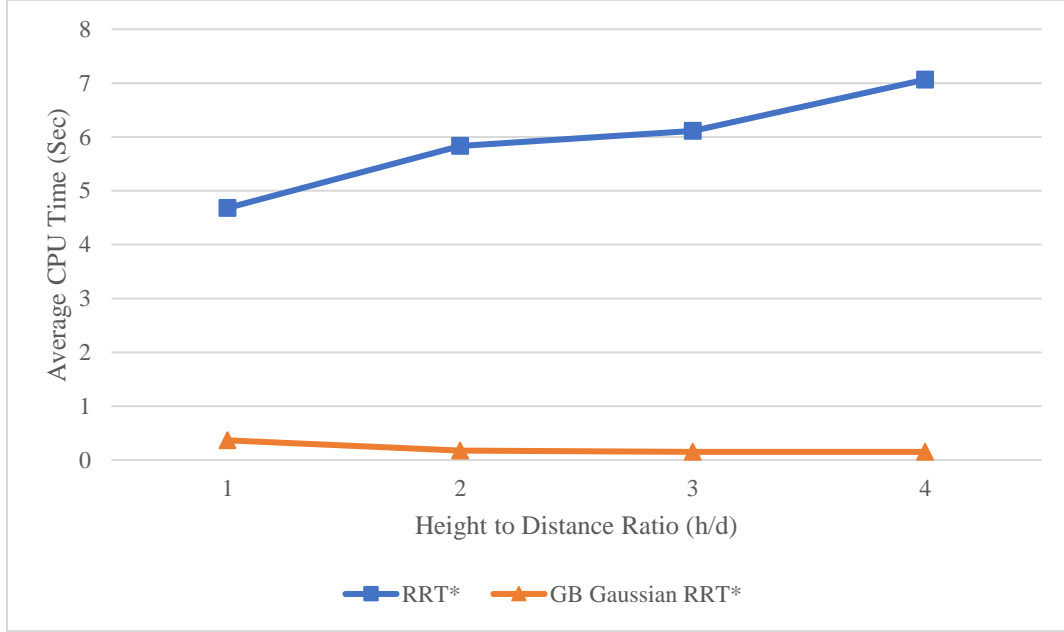


Fig. 24 Comparison Between Different Height to Distance Ratio vs Average CPU Time

Fig. 25 shows the comparison between optimum tolerance and the average time to perform the algorithms. For 3%, 2.5%, 2%, 1.5%, 1% optimum tolerance and for ratio  $h/d = 1$ , the average time for RRT\* is 3.543sec, 4.306sec, 4.762sec, 5.378sec and 6.345sec, respectively, and for the same map the proposed GB Gaussian RRT\* perform in average 0.226sec, 0.283sec, 0.336sec, 0.426sec, 0.657sec, respectively. For a more cost-effective path, RRT\* required more iterations which lead to more computational time.

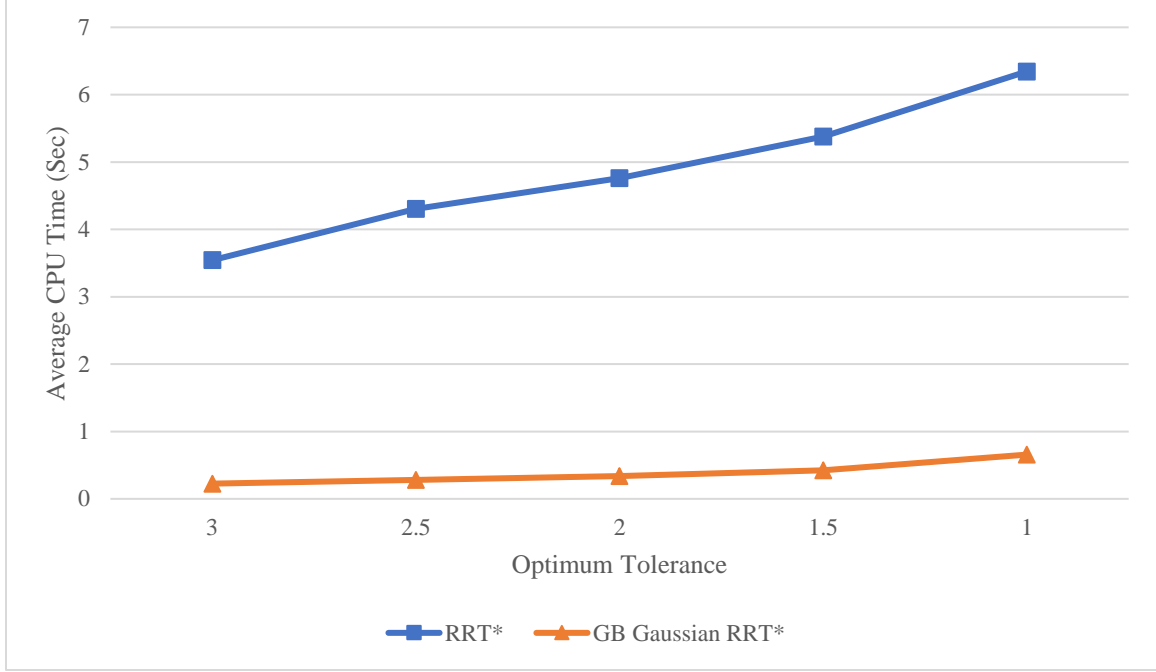


Fig. 25 Optimum Tolerance with Respect to the Cost vs Time to Run

#### 4.2 Map-2: Width Effect

Map 2 is a width effect experiment. For this test,  $x_{start} = (100,500)m$  and  $x_{end} = (900,500)m$  and the width of the obstacle  $w$  is changing with respect to the height of the state space. The height of the state space is fixed 1000m. As shown in Fig. 26 and Fig. 27, shadow of the obstacle is highlighted on respective axis. The coordinates of the obstacle corners are (400,400)m, (400,600)m, (600,600)m and (600,400)m. The RRT\* algorithm was used for map-2 to find the best path shown in Fig. 26, within 7.54sec and cost of 1018.88m. For Fig. 27 the proposed GB Gaussian RRT\* used, to find path and time taken was 0.737sec and cost for the path was 835.105m. The ideal cost for this map is 832.46m so the proposed GB Gaussian RRT\* is very close to ideal cost in very less time to perform, with compare to RRT\*.

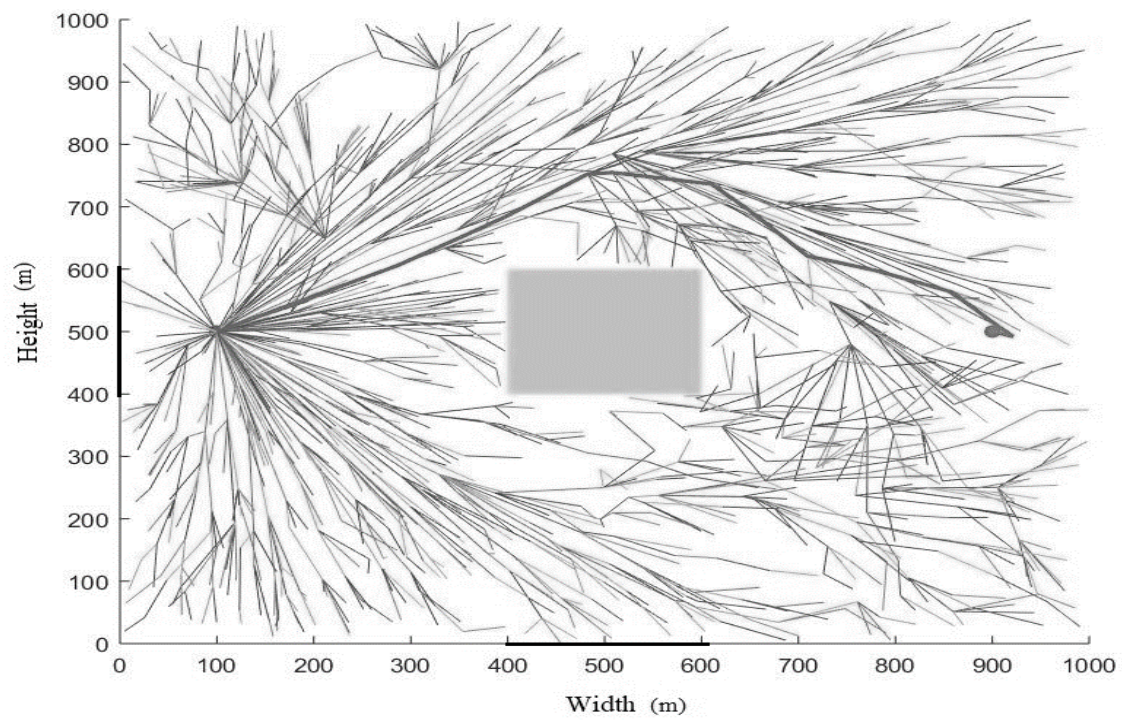


Fig. 26 Map-2 Solution Using RRT\*

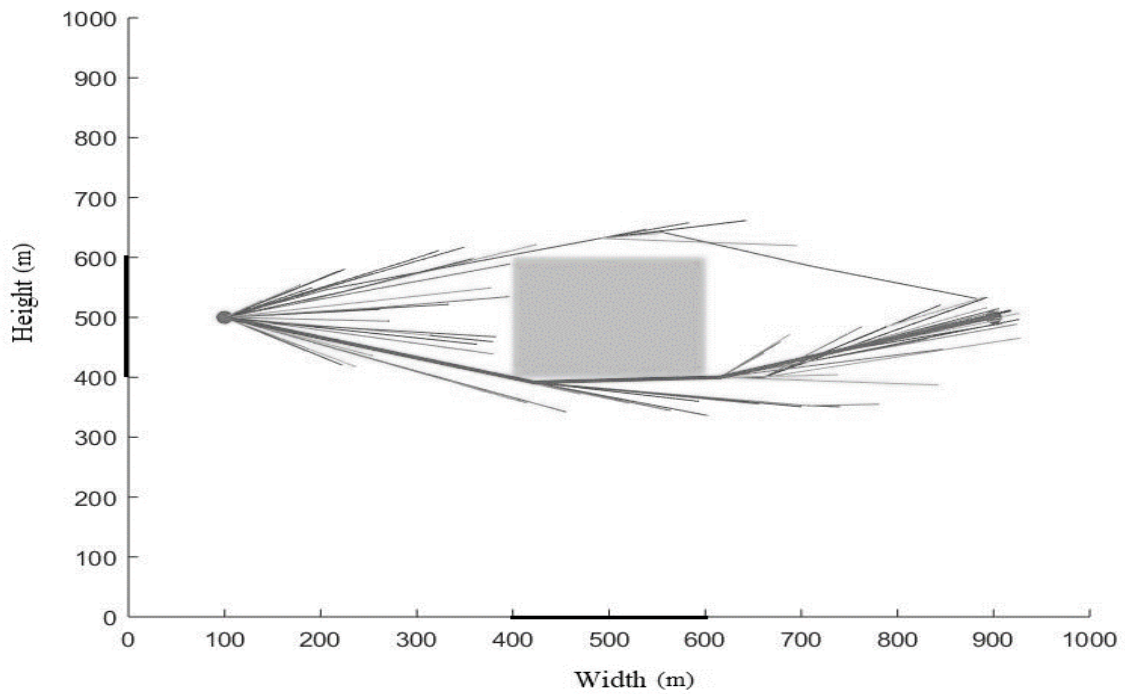


Fig. 27 Map-2 Solution Using the Proposed GB Gaussian RRT\*

Fig. 28 and Fig. 29 show the comparison between RRT\* and the proposed GB Gaussian RRT\* algorithm implementations on the width effect map with changing the width of the obstacle. The ratio of the height of state space to the width of an obstacle as shown by  $h/w$ . For ratios 10, 5, 3.3, 2.5, path planning using RRT\* required average time 5.988sec, 5.947sec, 6.284sec and 6.423sec, respectively, and the average distance cost for the path was 836.56m, 868.613m, 908.315m, 961.778m, respectively. For the same maps, the proposed GB Gaussian RRT\* took an average 0.471sec, 0.827sec, 1.095sec, 2.213sec, respectively, and the average cost for the path was 819.165m, 844.835m, 884.066m, and 935.645m while the ideal costs for all this ratios were 808.28m, 832.46m, 870.82m, 921.11m. As per the values the proposed GB Gaussian RRT\* provides very similar path costs, when compare to RRT\* with respect to ideal cost in very less amount of time.

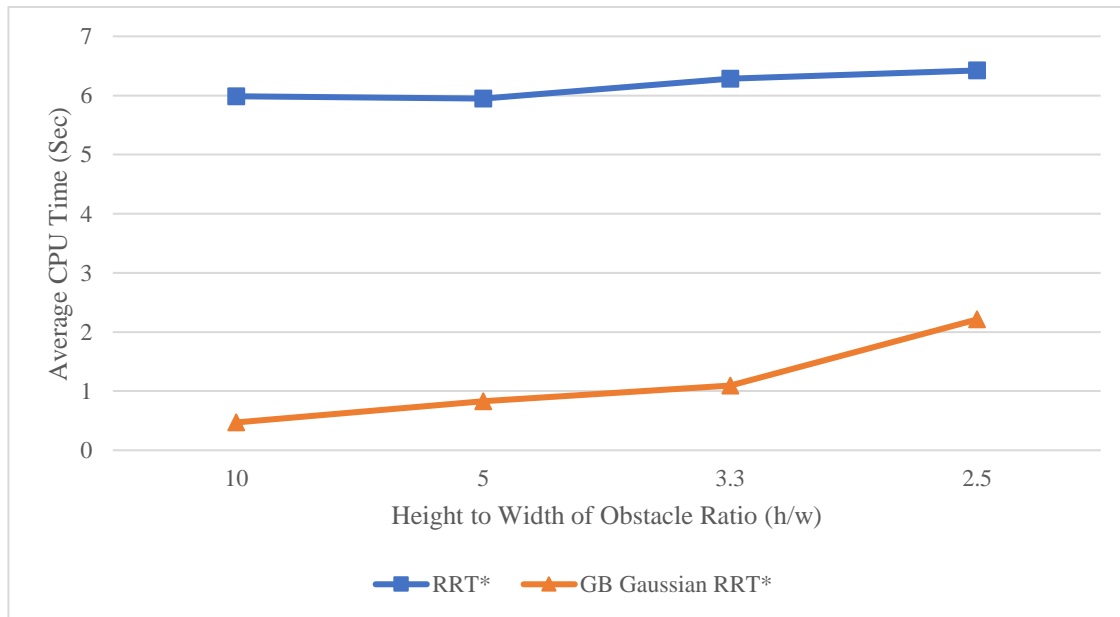


Fig. 28 Comparison Between Different Width Size to Average CPU Time

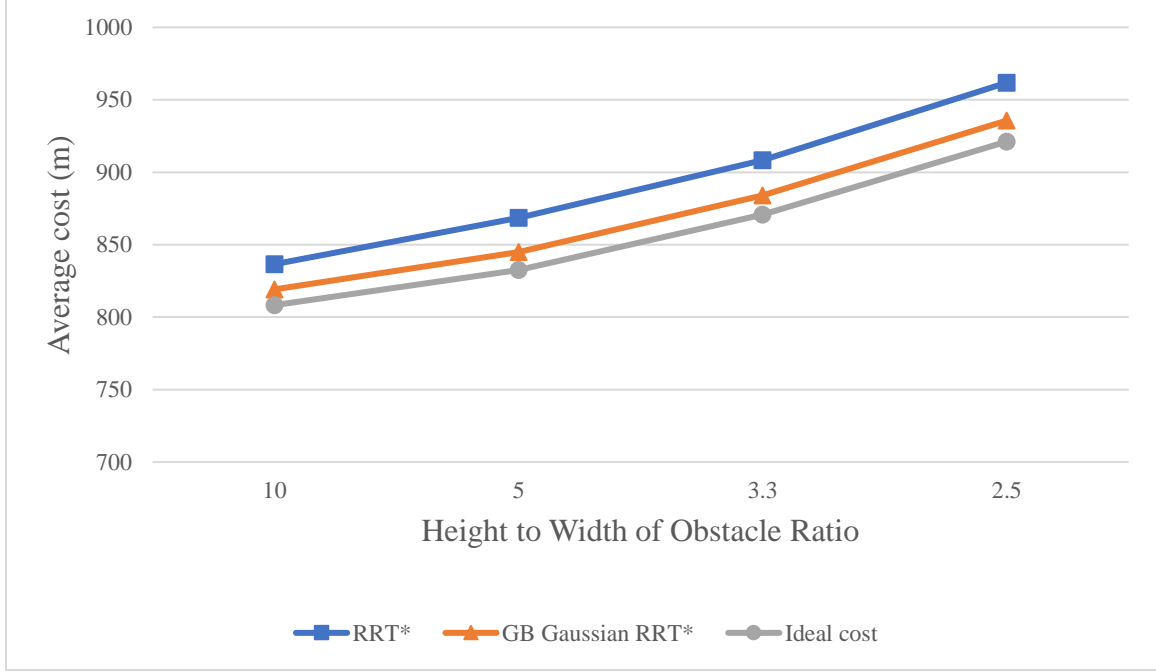


Fig. 29 Comparison Between Different Width Size to Average Cost

#### 4.3 Map-3: Gap Effect

A narrow gap is a critical problem for path planning. Map-3 shows the map for the narrow gap experiment. For this experiment,  $x_{start} = (100,500)m$ ,  $x_{end} = (900,500)m$  and as shown in Map-3,  $h_g$  is the size of the gap of an obstacle. As shown in Fig. 30 and Fig. 31, shadows of the obstacles are highlighted on respective axis. Coordinates for the obstacle below the gap are  $(400,300)m$ ,  $(400,480)m$ ,  $(600,480)m$  and  $(600,300)m$  and coordinates for the obstacle above gap are  $(400,520)m$ ,  $(400,700)m$ ,  $(600,700)m$  and  $(600,520)m$ .

Fig. 30 shows the implementation of RRT\* on the gap effect map. To find the near optimal path, the algorithm required 9.88sec and the cost of the path was 1027.35m. for the same map, Fig. 31 shows the implementation of the proposed GB Gaussian RRT\*, which took 0.359sec and the

cost of the path was 800.347m. As the ideal cost for this map is 800m, the proposed GB Gaussian RRT\* results are very close to the ideal path in very less amount of computational time.

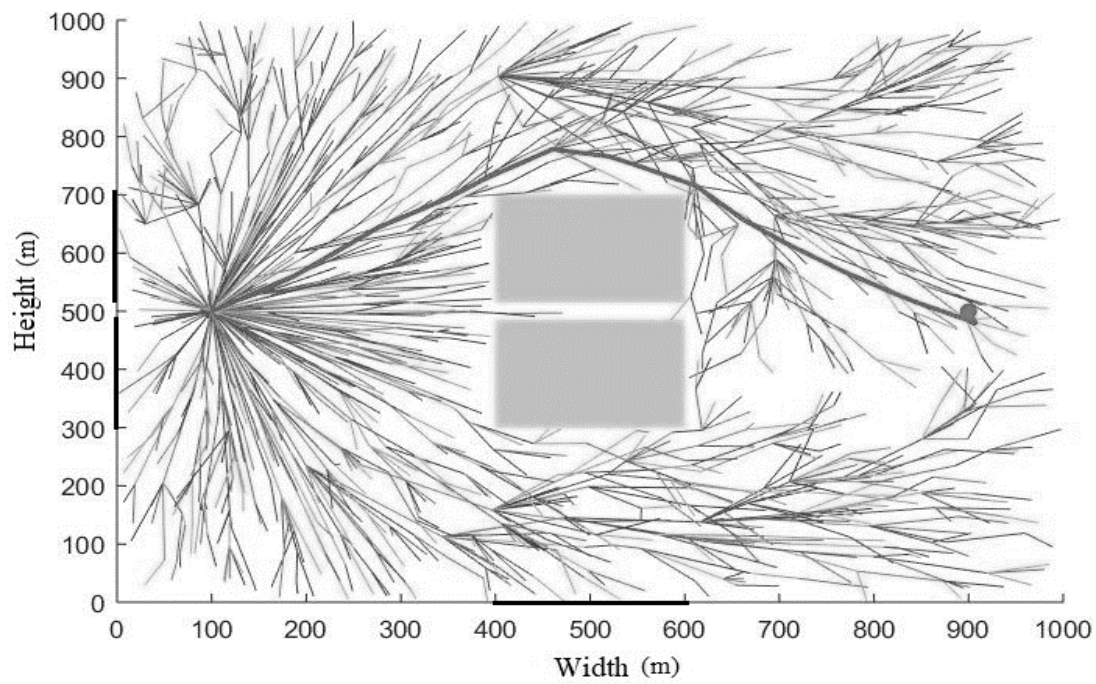


Fig. 30 Map-3 Solution Using the Proposed GB Gaussian RRT\*

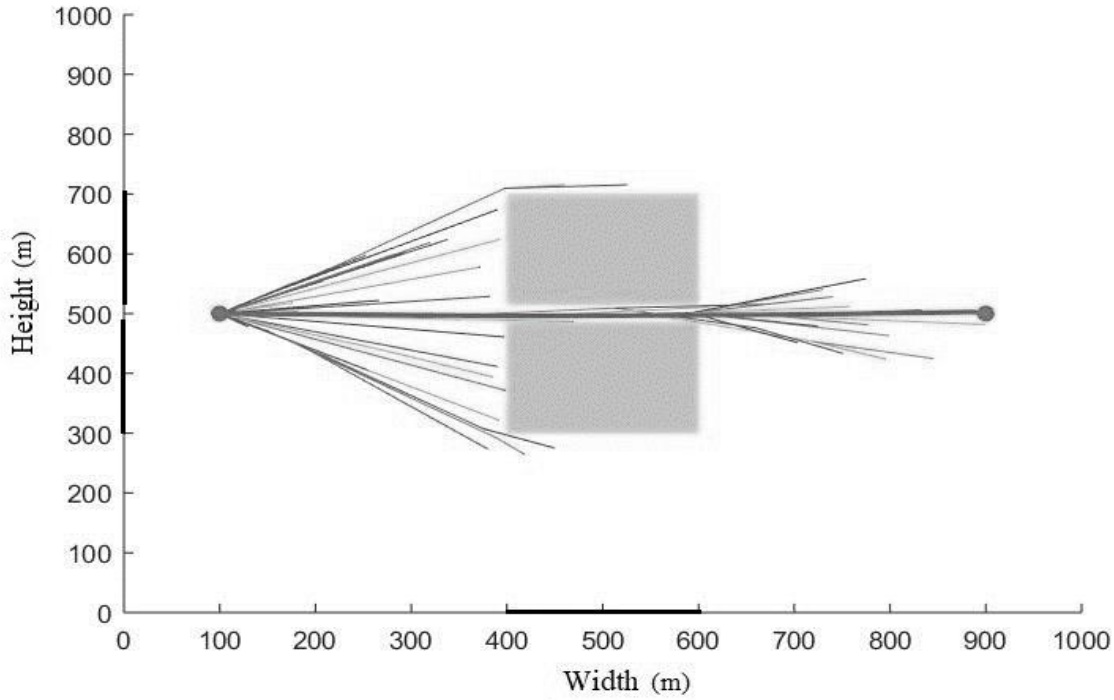


Fig. 31 Map-3 Solution Using the Proposed GB Gaussian RRT\*

Fig. 32 and Fig. 33 show the comparison for the RRT\* and the proposed GB Gaussian RRT\* algorithms for different gap sizes with respect to the height of the state space. For the experiments, 3%, 4%, 6%, 8% of height were taken as  $h_g$ . For all values of the  $h_g$  the RRT\* algorithm required the average time of 6.249sec, 6.252sec, 5.292sec, 5.645sec and average cost 895.501m, 890.229m, 852.832m, 846.26m, respectively. The proposed GB Gaussian algorithm was implemented for the same  $h_g$  values and resulted in average times of 2.586sec, 1.832sec, 0.932sec, 0.805sec and the average costs of 807.591m, 808.434m, 807.269m, 807.758m, respectively. Results show that most of the time RRT\* misses the narrow gap but the proposed GB Gaussian RRT\* successfully find the path from narrow gap most of the time.

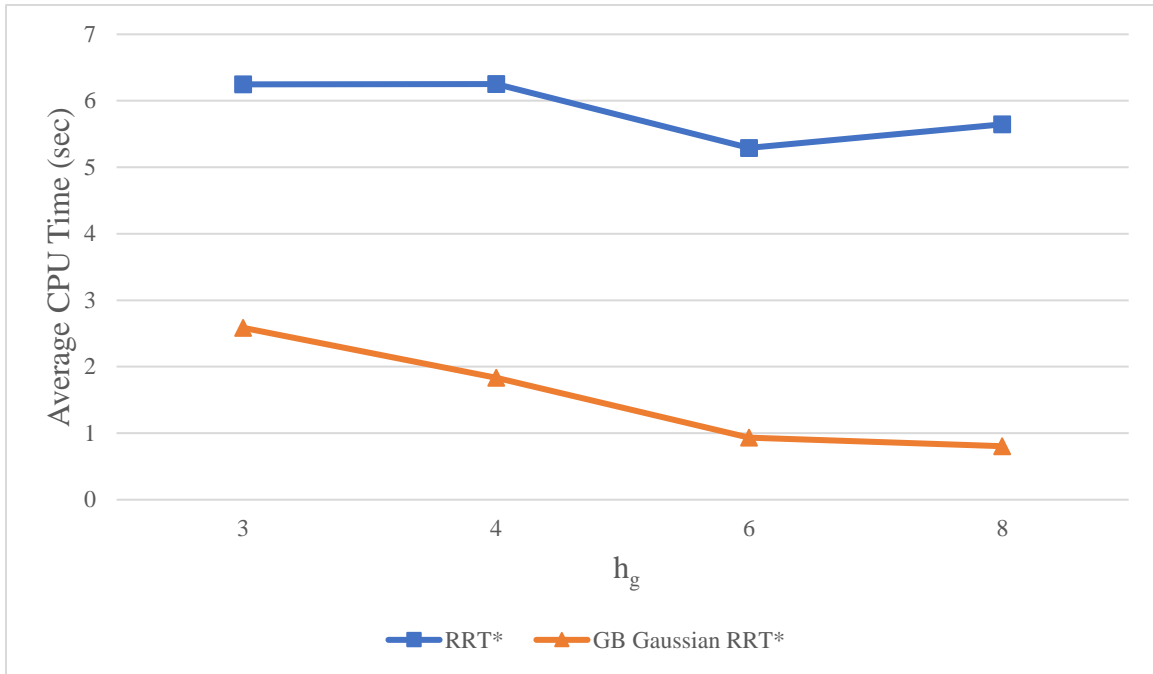


Fig. 32 Comparison Between Gap Size to Average CPU Time

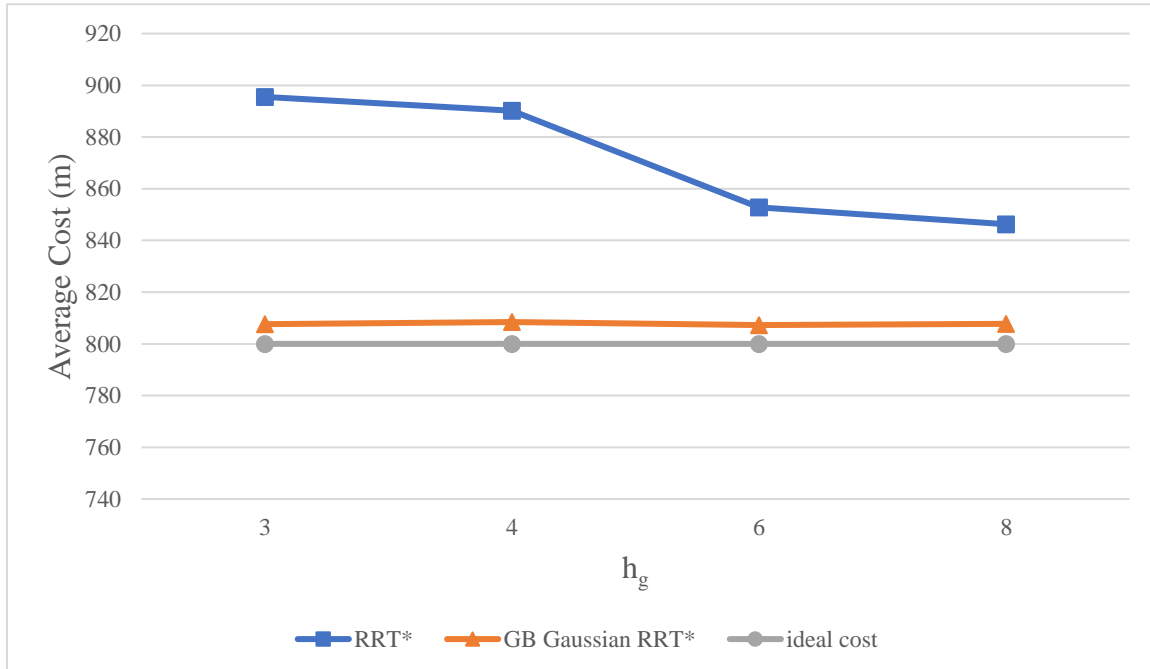


Fig. 33 Comparison Between Different Gap Size to Average Cost



#### 4.4 Map-4: Two Obstacle for Trap Check

Map-4 two pillars map is useful to find the algorithm compatibility in a complex environment for a successful operation of coming out of the trap. For this experiment,  $x_{\text{start}} = (100,700)\text{m}$  and  $x_{\text{end}} = (900,900)\text{m}$ , the ideal path cost is 1023.15m. As shown in, Fig. 34 and Fig. 35, shadows of the obstacles are highlighted on respective axis. Coordinates for the obstacle on left side are  $(300,500)\text{m}$ ,  $(300,1000)\text{m}$ ,  $(400,1000)\text{m}$  and  $(400,500)\text{m}$  and coordinates for the obstacle on right side are  $(700,300)\text{m}$ ,  $(700,700)\text{m}$ ,  $(800,700)\text{m}$  and  $(800,300)\text{m}$ . For this test, both algorithms find the path for 5% accuracy to the ideal path. Fig. 34 shows the implementation of RRT\* algorithm on trap check map algorithm that took 8.51sec and cost for the path was 1286.38m. For the same map the proposed GB Gaussian RRT\* took 0.68sec and cost of the path was 1042.58m shown in Fig. 35.

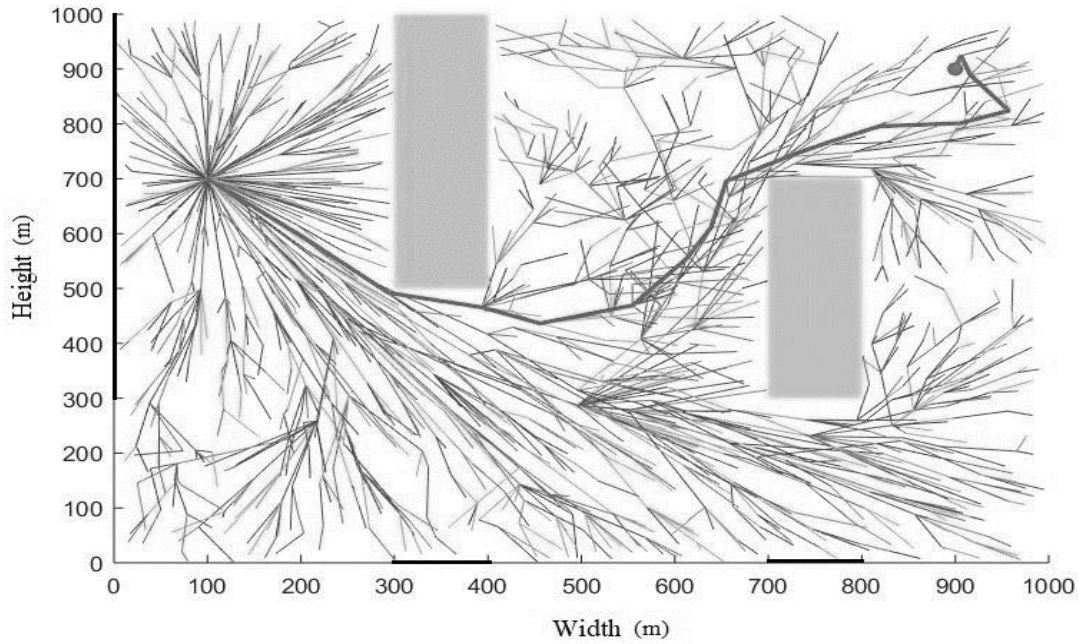


Fig. 34 Map-4 Solution Using RRT\*

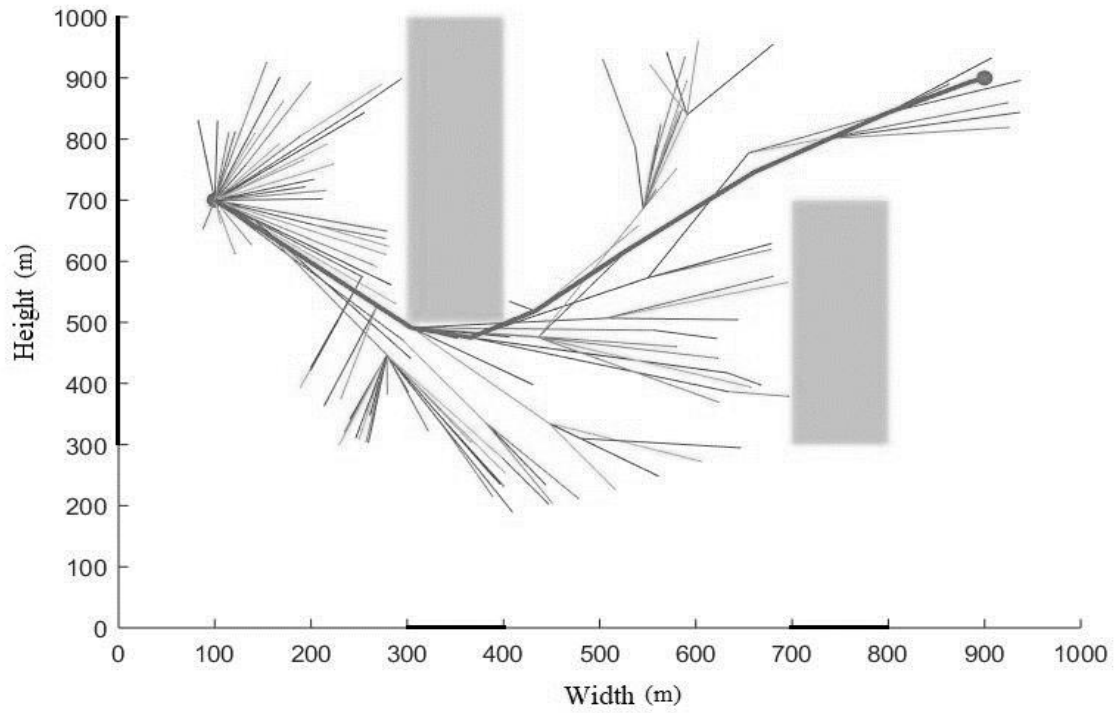


Fig. 35 Map-4 Solution Using the Proposed GB Gaussian RRT\*

Fig. 36 shows the results for the 100 runs for the trap check for both algorithms. For RRT\*, the average completion time was 6.005sec and the average cost was 1097.6m. for the proposed GB Gaussian RRT\*, the average completion time was 1.602sec and average cost 1064.35m. i.e., the proposed GB Gaussian RRT\* can provide the same cost path in very less time.

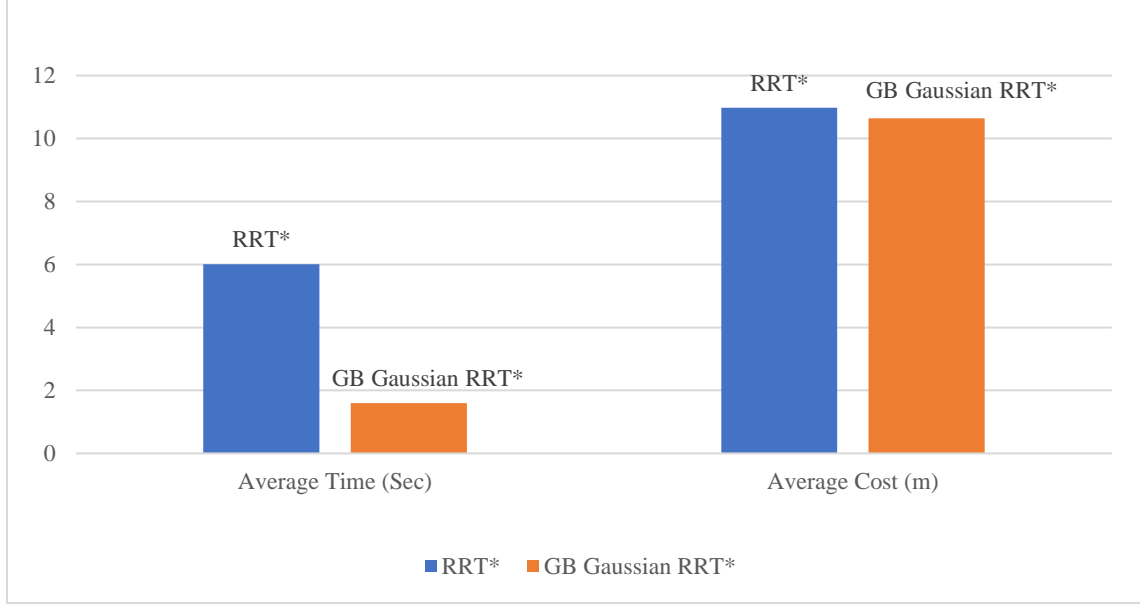


Fig. 36 Results of 100 Run for Trap Check Problem

#### 4.5 Map-5: T Type Obstacle

Map-5 shows the T shape obstacle map. T shape obstacle is critical in terms of the local minima. For this experiment,  $x_{start} = (100,500)m$ ,  $x_{end} = (900,500)m$  and the ideal cost of the path was 906.225 m. As shown in Fig. 37 and Fig. 38, shadows of the obstacles are highlighted on respective axis. Coordinates for the horizontal object are (300,600)m, (300,700)m, (700,700)m and (700,600)m and coordinates for vertical object are (450,300)m, (450,600)m, (550,600)m and (550,300)m. Fig. 37 shows the implementation of RRT\* algorithm for T shape obstacles with the completion time of 8.856 sec and cost of the path was 1059.16m. For the same map, as shown in Fig. 38, the proposed GB Gaussian RRT\* required 4.62 sec and cost for the path was 913.026m. As shown in Fig. 39, for the RRT\* average cost was 964.515m and the average time was 7.632sec and for the proposed GB Gaussian RRT\* average cost was 921.105m and average time was

3.123sec, i.e., the proposed GB Gaussian RRT\* provides the same cost path of RRT\* but almost at 50% the former computational time.

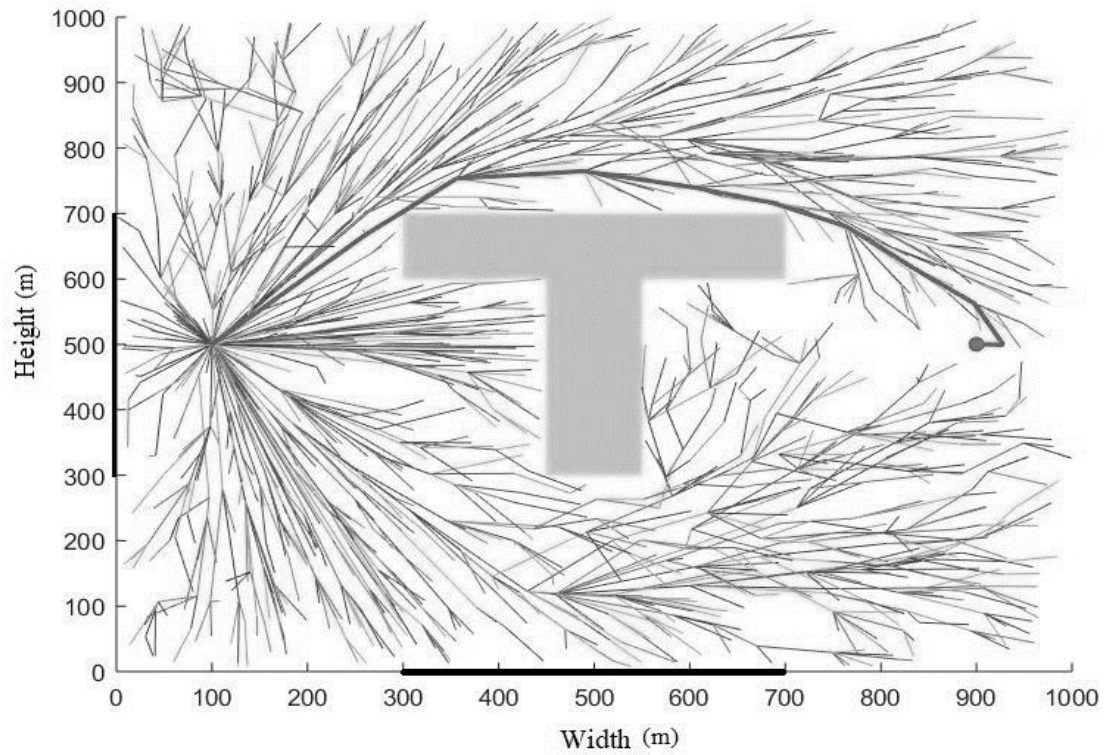


Fig. 37 Map-5 Solution With RRT\*

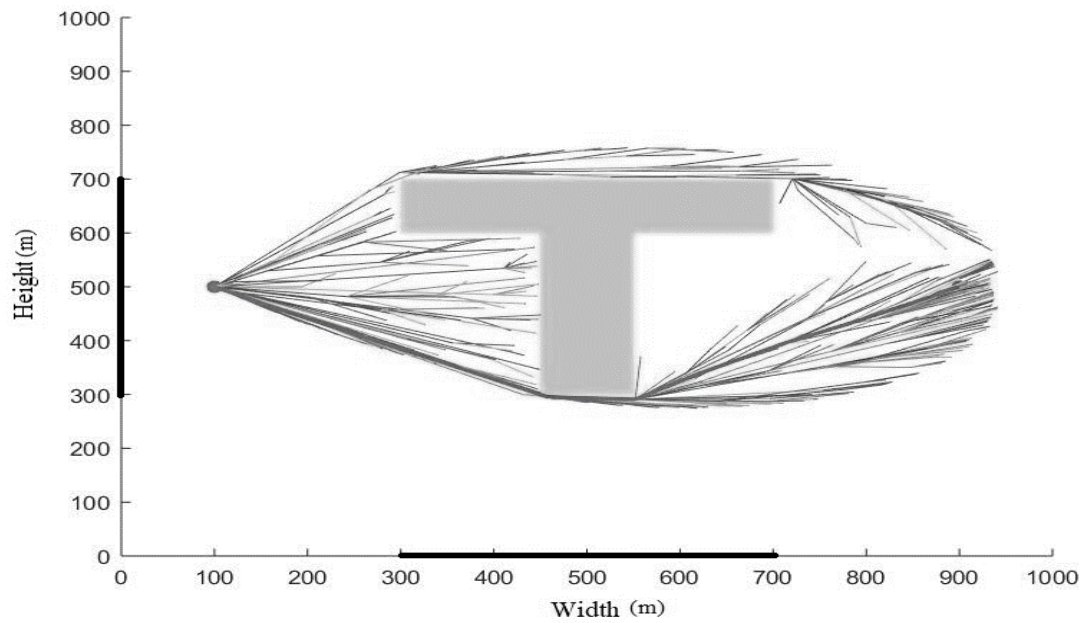


Fig. 38 Map-5 Solution with the Proposed GB Gaussian RRT\*

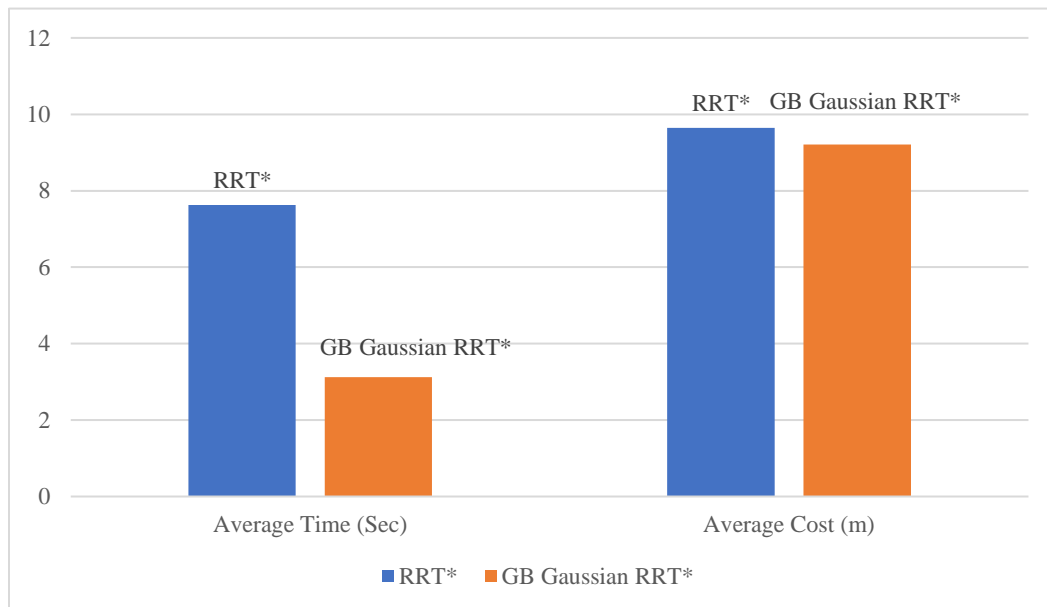


Fig. 39 Results for Map 5 with 100 Independent Runs

## CHAPTER 5

### CONCLUSIONS

This research introduces the proposed Goal Biased Gaussian RRT\* Algorithm to find the optimal path in the least amount of time. The algorithm can perform this task, because for the first path algorithm is biased to the goal and once the algorithm finds an initial path, using that data the algorithm generates new sampling points using Gaussian Distribution. The proposed Goal Biased Gaussian RRT\* outperforms RRT\* in all five different illustrate maps and provides the feasible path in the least amount of time. Research findings show that the proposed algorithm finds an asymptotically optimal path in ten times less computational time compared to RRT\* in no obstacle situations. Even the size of the obstacle does not make more difference in efficiency of a path length and computational time. For the narrow gap situation where most planners fail to find a path from the gap, the proposed Goal Biased Gaussian RRT\* efficiently finds a path within a gap. The numerical experiments conclude that the proposed Goal Biased Gaussian RRT\* is more efficient and fast method compared to RRT\*.

The proposed Goal Biased Gaussian RRT\* is an enhancement of the RRT\* so it can implement the system dynamics to generate a more realistic path. To implement this methodology on autonomous vehicles, more work must be done regarding the boundary value problem to generate feasible trajectories in the path. Also, the proposed Goal Biased Gaussian RRT\* can modify to find the path in real time with moving obstacles or obstacles with some uncertain boundary definitions.

## REFERENCES

- [1] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli and J. P. How, “Real-Time Motion Planning With Applications to Autonomous Urban Driving,” in *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105-1118, 2009.
- [2] Ying Liu and N. I. Badler, “Real-time reach planning for animated characters using hardware acceleration,” *Proceedings 11th IEEE International Workshop on Program Comprehension*, New Brunswick, NJ, USA, pp. 86-93, 2003.
- [3] J.-C. Latombe, “Motion planning: A journey of robots, molecules, digital actors, and other artifacts,” *The International Journal of Robotics Research*, vol. 18, no. 11, pp. 1119–1128, 1999.
- [4] D. González, J. Pérez, V. Milanés and F. Nashashibi, “A Review of Motion Planning Techniques for Automated Vehicles,” in *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135-1145, 2016.
- [5] F. Schler, “3d path planning for autonomous aerial vehicles in constrained spaces [Ph.D. thesis],” *Department of Electronic Systems, Faculty of Engineering and Science, Aalborg University*, Aalborg, Denmark, 2012.
- [6] H. M. Choset, “Sampling-Based Algorithms,” in *Principles of Robot Motion: Theory, Algorithms, and Implementations*, eds., Cambridge, Massachusetts, USA, MIT Press, Ch. 7, pp. 193-268, 2005.
- [7] S. M. LaValle, “Sampling-Based Motion Planning,” in *Planning Algorithms*, eds., Urbana, Illinois, USA, Cambridge University Press, Ch. 5, pp. 185-248, 2006.

- [8] A. Li Han, "Hybrid probabilistic RoadMap - Monte Carlo motion planning for closed chain systems with spherical joints," *IEEE International Conference on Robotics and Automation, Proceedings, ICRA '04*, New Orleans, LA, USA, vol. 1, pp. 920-926, 2004.
- [9] Chamseddine, Y. Zhang, C. A. Rabbath, C. Join and D. Theilliol, "Flatness-Based Trajectory Planning/Replanning for a Quadrotor Unmanned Aerial Vehicle," in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 4, pp. 2832-2848, 2012.
- [10] D. Meyer-Delius, M. Beinhofer, and W. Burgard, "Occupancy grid models for robot mapping in changing environments," in *Proceedings of the 20th AAAI Conference on Artificial Intelligence*, Toronto, Canada, 2012.
- [11] Y. B. Sebbane, "Mission Planning," in *Lighter Than Air Robots: Guidance and Control of Autonomous Airships*, vol. 58, Netherlands, Springer, Ch. 3, pp. 45-97, 2012.
- [12] D. Neel, M. S. Sakhalkar, S. Agarwal, D. P. Neel and S. Ray, "Routing algorithms with adaptive weight function based on total wavelengths and expected available wavelengths and frequency of usage in optical WDM networks," *2006 IFIP International Conference on Wireless and Optical Communications Networks*, Bangalore, p. 4, 2006.
- [13] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia, "Survey of Robot 3D Path Planning Algorithms," *Journal of Control Science and Engineering*, vol. 2016, p. 22, 2016.
- [14] M. Montemerlo, "Junior: The Stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569-597, 2008.
- [15] H. Pan, C. Guo, and Z. Wang, "Research for path planning based on improved astart algorithm," *2017 4th International Conference on Information, Cybernetics and Computational Social Systems (ICCSS)*, Dalian, pp. 225-230, 2017.



- [16] A. Stentz, "Optimal and efficient path planning for partially-known environments," *Proc. IEEE int. Conf. Robot. Autom.*, pp. 3310-3317, 1994.
- [17] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field  $D^*$  algorithm," *J. Field Robot.*, vol. 23, no. 2, pp. 79-101, 2006.
- [18] G. N. Varela and M. C. Sinclair, "Ant colony optimization for virtual-wavelength-path routing and wavelength allocation," *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, Washington, DC, p. 1816, vol. 3, 1999.
- [19] C. C. Hsu, W. Y. Wang, Y. H. Chien, R. Y. Hou and C. W. Tao, "FPGA implementation of improved ant colony optimization algorithm for path planning," *2016 IEEE Congress on Evolutionary Computation (CEC)*, Vancouver, BC, pp. 4516-4521, 2016.
- [20] S. M. LaValle, "Rapidly Exploring Random Tree – A new tool for path planning," in Tr 98-11, *computer science dept., Iowa State University*, 1998, <http://msl.cs.uiuc.edu/~lavalle/rrtpubs.html> [Accessed 19 November 2018].
- [21] K. Sertac, E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems*, Zaragoza, Spain, vol. 1, 2010.
- [22] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, vol. 2, pp. 1178-1183, 2003.
- [23] D. Kim, J. Lee and S. e. Yoon, "Cloud RRT\*: Sampling Cloud based RRT\*," *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, pp. 2519-2526, 2014.
- [24] D. Ferguson and A. Stentz, "Anytime RRTs," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, pp. 5369-5375, 2006.

- [25] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT\*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, pp. 2997-3004, 2014.
- [26] L. Palmieri and K. O. Arras, "Distance metric learning for RRT-based motion planning with constant-time inference," *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, pp. 637-643, 2015.
- [27] A. O. Dempster, A.N. Laird, and D.B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, pp.1-38, 1977.
- [28] M. Lavalley, Steven & Kuffner, James, "Rapidly-Exploring Random Trees: Progress and Prospects," in *Proceedings Workshop on the Algorithmic Foundations of Robotics*, 2000, <http://msl.cs.uiuc.edu/~lavalley/rrtpubs.html> [Accessed 19 November 2018].
- [29] Tor, Arne, Tor & Hanssen, "The multivariate normal inverse gaussian distribution: Em-estimation and analysis of synthetic aperture sonar data," *Alfred & Edgar Hansen, Roy*, 2018.
- [30] A. Chokniwal and M. Singh, "Faster Mahalanobis K-means clustering for Gaussian distributions," *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Jaipur, pp. 947-952, 2016.
- [31] E. Lughofer, "eVQ-AM: An extended dynamic version of evolving vector quantization," *IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, Singapore, pp. 40-47, 2013.
- [32] Christopher Bishop, "Mixtur Models and EM." in *Pattern Recognition and Machine Learning*, 1<sup>st</sup> ed., New York, NY, USA, *Springer-Verlag New York*, Ch. 9, pp. 423-460, 2006.

## APPENDIX I

### 7.1 MATLAB Code

```
%run RRT* and GB Gaussian RRT* algorithm multiple time to
%generate graph

clear;

%number of time run prog
numprog = 1;

format longG

%code parameters

height = 1000;
width = 1000;
qstart = [100,700];
qgoal = [900,900];
iterations = 1500;
obs1.x=[300 400];
obs1.y=[500 1000];
obs2.x = [700 800];
obs2.y = [300 700];
idealdist= eudist(qstart,qgoal);
widealdist = 1023.15;
ratio = 2.5;
opt = 5;
optimuntolarance = ((widealdist*opt)/100)+widealdist;

costforpath = zeros(numprog,1);
timetorun = zeros(numprog,1);
out = 0;
gbcostforpath = zeros(numprog,1);
gbtimetorun = zeros(numprog,1);
gbout = 0;

for k = 1:numprog

    % RRT*
```

```

        [costforpath(k),timetorun(k),out] =
RRT_star_final_mul(qstart,qgoal,height,width,iterations,obs
1,obs2,optimumtolarence,out);

end

for l = 1:numprog

    %GB RRT*
    [gbcostforpath(l),gbtimetorun(l),gbout] =
GB_Gau_RRT_final_mul(qstart,qgoal,height,width,iterations,o
bs1,obs2,optimumtolarence,idealdist,gbout);

end

% RRT* Data
avgcost = mean(costforpath);
avgtime = mean(timetorun);
[bestpath,index] = min(costforpath);
[worstpath,worstindex] = max(costforpath);
timebestpath = timetorun(index);
timeworstpath = timetorun(worstindex);
%display
fprintf('no obstacle RRT* h/d=%G optimumtolarence = %G,value
= %G \n',ratio,opt,optimumtolarence)
fprintf('Eucleadian distance between start to goal=
%G\n',idealdist)
fprintf('ideal map cost between start to goal=
%G\n\n',widealdist)
fprintf('Highest path = %G\n',worstpath)
fprintf('Highest path index = %G\n',worstindex)
fprintf('Time to find Highest path = %G\n\n',timeworstpath)
fprintf('Best path = %G\n',bestpath)
fprintf('Best path index = %G\n',index)
fprintf('Time to find Best path = %G\n\n',timebestpath)
fprintf('Average cost= %G\n',avgcost)
fprintf('average time to run= %G\n',avgtime)
fprintf('over iterations = %d\n\n',out)

% GB Gaussian RRT* Data
gbavgcost = mean(gbcostforpath);
gbavgtime = mean(gbtimetorun);

```

```

[gbbestpath,gbindex] = min(gbcostforpath);
[gbworstpath,gbworstindex] = max(gbcostforpath);
gbtimebestpath = gbtimetorun(gbindex);
gbtimeworstpath = gbtimetorun(gbworstindex);
%display
fprintf('no obstacle GB Gaussian RRT* h/d=%G optimumtolarence
= %G,value = %G\n',ratio,opt,optimumtolarence)
fprintf('Eucleadian distance between start to goal=
%G\n',idealdist)
fprintf('ideal map cost between start to goal=
%G\n\n',widealdist)
fprintf('Highest path = %G\n',gbworstpath)
fprintf('Highest path index = %G\n',gbworstindex)
fprintf('Time to find Highest path = %G\n\n',gbtimeworstpath)
fprintf('Best path = %G\n',gbbestpath)
fprintf('Best path index = %G\n',gbindex)
fprintf('Time to find Best path = %G\n\n',gbtimebestpath)
fprintf('Average cost= %G\n',gbavgcost)
fprintf('average time to run= %G\n',gbavgtime)
fprintf('over iterations = %d\n',gbout)

```

```

% code for GB Gaussian RRT*
function [finalcostofpath,timetorun,out] =
GB_Gau_RRT_final_mul(qstart,qgoal,height,width,iterations,o
bs1,obs2,optimumtolarence,idealdist,out)

```

```

close(findobj('type','figure','name','RRT basic'));
close(findobj('type','figure','name','RRT growing'));
format longG
%%
% define boundry and area.

```

```

figure('name', 'GB Gaussian RRT star');
axis ([0 width 0 height]);
hold on
slop = abs(qgoal(2)-qstart(2))/(qgoal(1)-qstart(1));
qcenter = [qstart(1)+(((qgoal(1)-
qstart(1))/idealdist)*(idealdist/2)),qstart(2)+(((qgoal(2)-
qstart(2))/idealdist)*(idealdist/2))];

```

```

%define OBSTACLE

%create obstacle1
x1box=obs1.x([1 1 2 2 1]);
y1box=obs1.y([1 2 2 1 1]);

%create obstacle2
x2box=obs2.x([1 1 2 2 1]);
y2box=obs2.y([1 2 2 1 1]);

obs1area      =      abs(obs1.x(1)-obs1.x(2))*abs(obs1.y(1)-
obs1.y(2));
obs2area      =      abs(obs2.x(1)-obs2.x(2))*abs(obs2.y(1)-
obs2.y(2));

totalarea = height*width;
maxfree = abs(totalarea - (obs1area+obs2area));
ddimension = 2;
unitballvolume = 2*pi;

% define tree edge, node and vertice
%
% vertecies: An array of Vertrcies' coordinates, sorted by
the generated
% order

vertecies = qstart;
vert_count = 1;
qtree(vert_count,:)=num2cell(vert_count,[1 2]);
sol_tree_count = 1;
sol_tree_log(sol_tree_count,:) =num2cell(sol_tree_count,[1
2]);
sol_tree_cost(sol_tree_count,:) = 0;
%vertecies.cost = 0;
eps=(height*3)/100;
eps1=(height*5)/100;
val = 0;
cmin(vert_count,:)=0;
solcost = 0;
soltree = qstart;
qnew_cost = 0;
cmax = 0;

%

```

```

% Edges:starting and ending of each edges, note edges(i)
corresponds to vertecies(i+1),
% because vertecies(1) is the original point

edges.x = zeros(iterations,2);
edges.y = zeros(iterations,2);
edge_count = 0;
%%
%
%
% ind_nearest: index to the nearest point.
vertecies(ind_nearest(i)) is the
% closetest vertex to vertecies(i+1)
%%
ind_nearest = zeros(iterations,1);
qnew = [0 0];
colli = 0;
i=1;
randcount=1;
colli_random1 = 0;

tic;
%%
while i <= iterations

    % sampling points generation
    [x_rand,y_rand,eps,eps1] =
samplepoint(height,width,cmax,idealdist,slop,qcenter,eps,eps1);

    temp.rand= [x_rand,y_rand]; %save temporary random point

    % find nearest point using KD-Tree method
    ind_nearest(i) = knnsearch(vertecies, [x_rand,y_rand]);
    qnearest =
[vertecies(ind_nearest(i),1),vertecies(ind_nearest(i),2)];

    qnew_cost1 = eudist(qnearest,[x_rand,y_rand]);
    qnew_goal_cost = eudist(qgoal,qnearest);
    %find new point using steer funcrtion or BVP
    %find new point towards goal if no obstacle eps1>eps
    %if there is obstacle eps>eps1
    %else use only eps

```

```

    %A goabbiasing function
    %check collision

    if colli_random1 == 0
        colli_random =
collision(x1box,y1box,x2box,y2box,[x_rand,y_rand],qnearest)
;
        if colli_random == 0
            %this is tem new point
            qnew =
gbsteerfn(temp.rand(1),temp.rand(2),qnearest(1),qnearest(2)
,qnew_cost1,qnew_goal_cost,qgoal,eps,eps1);
            colli =
collision(x1box,y1box,x2box,y2box,qnew,qnearest);
            if colli == 1
                % value of eps and eps1 are inter change to
not stay in local minima
                qnew =
gbsteerfn(temp.rand(1),temp.rand(2),qnearest(1),qnearest(2)
,qnew_cost1,qnew_goal_cost,qgoal,eps1,eps);
                colli1 =
collision(x1box,y1box,x2box,y2box,qnew,qnearest);
                if colli1 == 1
                    %use normal steer function
                    qnew =
steerfn(temp.rand(1),temp.rand(2),qnearest(1),qnearest(2),q
new_cost1,eps1);
                    colli2 =
collision(x1box,y1box,x2box,y2box,qnew,qnearest);
                    if i<=30
                        colli_random1 =0;
                    end
                    if colli2 == 1

                        continue
                    end
                end
            end
        else

            qnew =
steerfn(temp.rand(1),temp.rand(2),qnearest(1),qnearest(2),q
new_cost1,eps1);

```



```

        colli3
collision(x1box,y1box,x2box,y2box,qnew,qnearest);
        if colli3 == 1
            continue
        end
    else
        qnew
gbsteerfn(temp.rand(1),temp.rand(2),qnearest(1),qnearest(2)
,qnew_cost1,qnew_goal_cost,qgoal,eps1,eps);
        colli3
collision(x1box,y1box,x2box,y2box,qnew,qnearest);
        if colli3 == 1
            continue
        end
    end

    if i == 150
        colli_random1 = 0;
    end

    if qnew(1) >= height || qnew(2) >= width
        continue
    end

    %calculate radius for rangesearch
    Rrrt
2*((1+(1/ddimension))^(1/ddimension))*((maxfree/unitballvol
ume)^(1/ddimension));
    dynamicradius
Rrrt*((log(vert_count)/vert_count)^(1/ddimension));
    %optimization in edge developement

    qnew_cost = eudist(qnearest,qnew);
    %try1 = abs((dynamicradius-eps)/2);

    qrangel=rangesearch(vertecies,qnew,max(dynamicradius,eps));
    %qrangel=rangesearch(vertecies,qnew,try1);
    qrangle=cell2mat(qrangel);

```

```

    cmin1 = cmin(ind_nearest(i)) + qnew_cost;
    qmin1 = qnearest;

    %rewire process
    for k = 1:length(qrange)
        cnew = cmin(qrange(k) +
eudist([verticies(qrange(k),1),verticies(qrange(k),2)],qnew
));
        if cnew < cmin1
            if 0 ==
collision(x1box,y1box,x2box,y2box,qnew,[verticies(qrange(k)
,1),verticies(qrange(k),2)])
                qmin1 =
[verticies(qrange(k),1),verticies(qrange(k),2)];
                cmin1 = cnew;
                ind_nearest(i) = qrange(k);
            end
        end
    end

    %verticies(vert_count+1,:) = [x_rand, y_rand];
    verticies(vert_count+1,:) = qnew;

    edges.x(edge_count+1,:) = [qmin1(1), qnew(1)];
    edges.y(edge_count+1,:) = [qmin1(2), qnew(2)];
    edge_count = edge_count + 1;
    qtree(vert_count+1,:) =
addtreenode(vert_count,qtree,ind_nearest(i));
    cmin(vert_count+1,:) = cmin1;

    for j = 1:length(qrange)
        cnear = cmin(qrange(j));
        cmin2 = cmin1 +
eudist(qnew,[verticies(qrange(j),1),verticies(qrange(j),2)]
));
        if cmin2 < cnear
            if 0 ==
collision(x1box,y1box,x2box,y2box,qnew,[verticies(qrange(j)
,1),verticies(qrange(j),2)])
                % ind_nearest(i) = vert_count+1;
                qnewrewire =
[verticies(qrange(j),1),verticies(qrange(j),2)];

```

```

        %vert_count_rewire = qrange(j);
        qtree(qrange(j),:) = addtreenode(qrange(j)-
1,qtree,vert_count+1);
        cmin(qrange(j),:) = cmin2;

    end
end
end

% find final path

goalcost = eudist (qnew,qgoal);

if goalcost <= eps
    solcost = cmin(vert_count+1,:) + goalcost;
    soltree =
createfinalpath(vertcies,qtree,vert_count,qgoal);
    %final_tree_count = 1;
    sol_tree_log(sol_tree_count,:) = num2cell(soltree,[1
2]);
    sol_tree_cost(sol_tree_count,:) = solcost;
    sol_tree_count = sol_tree_count + 1;

    [min_cost_tree,min_cost_index] = min(sol_tree_cost);
    soltree = cell2mat(sol_tree_log(min_cost_index));
    cmax= min_cost_tree;
    % break;

    if cmax <= optimumtolarence
        %disp(optimumtolarence)
        break;
    end
end

vert_count = vert_count + 1;
i=i+1;
end

```

```

%if not find final route

if cmax == 0
    out=out+1;
    ind_nearest1 = knnsearch(verticies,qgoal);
    qnearest =
[verticies(ind_nearest1,1),verticies(ind_nearest1,2)];
    finalcolli =
collision(x1box,y1box,x2box,y2box,qgoal,qnearest);
    if finalcolli == 0

        goalcost = eudist (qnearest,qgoal);
        cmax = cmin(ind_nearest1,:) + goalcost;
        soltree = createfinalpath(verticies,qtrees,ind_nearest1-
1,qgoal);
    else
        disp('no nearby points')
    end
end

%disp(cmax)

timetorun=toc;
finalcostofpath = cmax;
clear i x_rand y_rand edge_rand

hold on
mapshow(x1box,y1box,'DisplayType','polygon','LineStyle','none');
mapshow(x2box,y2box,'DisplayType','polygon','LineStyle','none');
scatter(qstart(1), qstart(2), 45, 'o','r','filled'); hold on;
scatter(qgoal(1), qgoal(2), 45, 'o','r','filled'); hold on;
scatter(verticies(:,1), verticies(:,2),
1,linspace(1,10,length(verticies(:,1))), 'filled'); hold on;
plot(edges.x', edges.y', 'LineWidth',0.01);
line(soltree(:,1),soltree(:,2), 'LineWidth',2, 'Color', 'r');

```

```
end
```

```
% Code for RRT*
function [finalcostofpath,timetorun,out] =
RRT_star_final_mul(qstart,qgoal,height,width,iterations,obs
1,obs2,optimumtolarance,out)
%clear;
%clc;
close(findobj('type','figure','name','RRT basic'));
close(findobj('type','figure','name','RRT growing'));
```

```
figure('name', 'RRT star');
axis ([0 width 0 height]);
hold on
```

```
% define starting and end point and iteration
```

```
%create obstacle1
x1box=obs1.x([1 1 2 2 1]);
y1box=obs1.y([1 2 2 1 1]);
```

```
%create obstacle2
x2box=obs2.x([1 1 2 2 1]);
y2box=obs2.y([1 2 2 1 1]);
```

```
obs1area = abs(obs1.x(1)-obs1.x(2))*abs(obs1.y(1)-
obs1.y(2));
obs2area = abs(obs2.x(1)-obs2.x(2))*abs(obs2.y(1)-
obs2.y(2));
```

```
totalarea = height*width;
maxfree = abs(totalarea - (obs1area+obs2area));
ddimension = 2;
unitballvolume = 2*pi;
```

```
% define tree edge, node and vertice
%
% vertecies: An array of Vertrcies' coordinates, sorted by
the generated
% order
```

```

verticies = qstart;
vert_count = 1;
qtree(vert_count,:)=num2cell(vert_count,[1 2]);
%verticies.cost = 0;
format longG

eps=(height*3)/100;
val = 0;
cmin(vert_count,:)=0;
finalcost = 0;
finaltree = qstart;
qnew_cost = 0;
%
%
% Edges:starting and ending of each edges, note edges(i)
% corresponds to verticies(i+1),
% because verticies(1) is the original point

edges.x = zeros(iterations,2);
edges.y = zeros(iterations,2);
edge_count = 0;
%%
%
%
%   ind_nearest:   index   to   the   nearest   point.
%   verticies(ind_nearest(i)) is the
%   closetest vertex to verticies(i+1)
%%
ind_nearest = zeros(iterations,1);
qnew = [0 0];
colli = 0;
i=1;
tic;
%%
while i <= iterations

    x_rand = width*rand();    % random point generation
    y_rand = height*rand();
    temp.rand= [x_rand,y_rand]; %save temporary random point


    % find nearest point using KD-Tree method

```

```

    ind_nearest(i) = knnsearch(verticies, [x_rand,y_rand]);
    qnearest =
[verticies(ind_nearest(i),1),verticies(ind_nearest(i),2)];

    qnew_cost1 = eudist(qnearest,[x_rand,y_rand]);
    %find new point using steer funcrtion or BVP
    qnew =
steerfn(temp.rand(1),temp.rand(2),qnearest(1),qnearest(2),q
new_cost1,eps); %this is tem new point

    %check collision
    colli =
collision(x1box,y1box,x2box,y2box,qnew,qnearest);

    if colli == 1
        continue
    end
    if qnew(1) >= height || qnew(2) >= width
        continue
    end
    %calculate radius for rangesearch
    Rrrt =
2*((1+(1/ddimension))^(1/ddimension))*((maxfree/unitballvol
ume)^(1/ddimension));
    dynamicradius =
Rrrt*((log(vert_count)/vert_count)^(1/ddimension));
    %optimization in edge developement

    qnew_cost = eudist(qnearest,qnew);

    qrange1=rangesearch(verticies,qnew,max(dynamicradius,eps));
    qrange=cell2mat(qrange1);

    cmin1 = cmin(ind_nearest(i)) + qnew_cost;
    qmin1 = qnearest;

    %rewire process
    for k = 1:length(qrange)

        cnew = cmin(qrange(k)) +
eudist([verticies(qrange(k),1),verticies(qrange(k),2)],qnew
);
        if cnew < cmin1

```

```

                                if                                0                                ==
collision(x1box,y1box,x2box,y2box,qnew,[vertecies(qrange(k)
,1),vertecies(qrange(k),2)])
                                qmin1                                =
[vertecies(qrange(k),1),vertecies(qrange(k),2)];
                                cmin1 = cnew;
                                ind_nearest(i) = qrange(k);
                                end
                                end
                                end

%vertecies(vert_count+1,:) = [x_rand, y_rand];
vertecies(vert_count+1,:) = qnew;

edges.x(edge_count+1,:) = [qmin1(1), qnew(1)];
edges.y(edge_count+1,:) = [qmin1(2), qnew(2)];
edge_count = edge_count + 1;
qtree(vert_count+1,:)                                =
addtreenode(vert_count,qtree,ind_nearest(i));
cmin(vert_count+1,:) = cmin1;

for j = 1:length(qrange)
    cnear = cmin(qrange(j));
    cmin2                                =                                cmin1                                +
eudist(qnew,[vertecies(qrange(j),1),vertecies(qrange(j),2)]
);
    if cmin2 < cnear
        if                                0                                ==
collision(x1box,y1box,x2box,y2box,qnew,[vertecies(qrange(j)
,1),vertecies(qrange(j),2)])
                                qnewrewire                                =
[vertecies(qrange(j),1),vertecies(qrange(j),2)];

                                qtree(qrange(j),:) = addtreenode(qrange(j)-
1,qtree,vert_count+1);
                                cmin(qrange(j),:) = cmin2;
                                end
                                end
                                end
end

```



```

% find final path

goalcost = eudist (qnew,qgoal);

if goalcost <= eps
    finalcost = cmin(vert_count+1,:) + goalcost;
    finaltree
createfinalpath(verticies,qtrees,vert_count,qgoal);
    %break;
    if finalcost <= optimumtolerance
        %disp(optimumtolerance)
        break;
    end
end

vert_count = vert_count + 1;
i=i+1;

end
%disp(randcount)

%if not find final route
if finalcost == 0
    out = out+1;
    ind_nearest1 = knnsearch(verticies,qgoal);
    qnearest
[verticies(ind_nearest1,1),verticies(ind_nearest1,2)];
    finalcolli
collision(x1box,y1box,x2box,y2box,qgoal,qnearest);
    if finalcolli == 0

        goalcost = eudist (qnearest,qgoal);
        finalcost = cmin(ind_nearest1,:) + goalcost;
        finaltree
createfinalpath(verticies,qtrees,ind_nearest1-1,qgoal);
    else
        disp('no nearby points')
    end
end
end
finalcostofpath=finalcost;

```

```

timetorun=toc;
clear i x_rand y_rand edge_rand

%disp(testidealdist)

hold on
mapshow(xlbox,ylbox,'DisplayType','polygon','LineStyle','none');
mapshow(x2box,y2box,'DisplayType','polygon','LineStyle','none');
scatter(qstart(1), qstart(2), 45, 'o','r','filled');
scatter(qgoal(1), qgoal(2), 45, 'o','r','filled');
scatter(verticies(:,1), verticies(:,2), 0.1, linspace(1,10,length(verticies(:,1))), 'filled');
plot(edges.x', edges.y');
line(finalltree(:,1),finaltree(:,2),'LineWidth',2,'Color','r');
end

```

```

function [xran,yran,eps,eps1] =
samplepoint(hi,wi,cmax,cid,slop,qcen,eps,eps1)
%create sampling point based on the first path
% use the initial path value to generate normal
distribution(gaussian distribution)
% so that limit the state space which help to get more cost
effective path
% in less time

```

```

if cmax >= 1
    eps=(cmax*3)/100;
    eps1=(cmax*5)/100;
    rotation = [cos(slop) sin(slop);sin(slop) cos(slop)];
    trandia = cmax/2;
    conjudia = (((((cmax)^2)-((cid)^2))^(1/2)))/2);
    ran = randn(1,2); %create random point
    ran1 = ran*[trandia 0;0 conjudia]; %calculate with

```

```

    ran2 = ran1*rotation;
    ran3 = ran2+qcen;
    xran = ran3(1);
    yran = ran3(2);

else
    xran = wi*rand();      % random point generation
    yran = hi*rand();
end

function a1=steerfn(x_rand,y_rand,qn1,qn2,qnc,eps)
    N = [0 0];
    % N1=[temp.rand;temp.nearest
    %find step size towards qrandom point
    if qnc>=eps
        %temp1= dist(temp.rand(1),temp.nearest(1));
        N(1) = abs(qn1 + ((x_rand-qn1)*eps)/qnc);
        N(2) = abs(qn2 + ((y_rand-qn2)*eps)/qnc);
    else
        N(1)=x_rand;
        N(2)=y_rand;
    end
    a1= [N(1),N(2)];
end

function
a1=gbsteerfn(x_rand,y_rand,qn1,qn2,qnc,qngc,xgoal,ep,ep1)
    N = [0 0];
    xgoal_x = xgoal(1);
    xgoal_y = xgoal(2);
    % N1=[temp.rand;temp.nearest
    %find step size towards qgoal point
    if qnc>=eps
        %temp1= dist(temp.rand(1),temp.nearest(1));
        % a = x_rand-qn1;
        % a2 = y_rand-qn2;
        N(1) = abs(qn1 + (((x_rand-qn1)/qnc)*ep)+(((xgoal_x-
qn1)/qngc)*ep1));
        N(2) = abs(qn2 + (((y_rand-qn2)/qnc)*ep)+(((xgoal_y-
qn2)/qngc)*ep1));

```

```

        else
            N(1)=x_rand;
            N(2)=y_rand;
        end
        a1= [N(1),N(2)];
    end

function d1 = eudist( dist1,dist2 )
% Find Euclidean Distance between two points in cartesian
system

d0=[dist1;dist2];
d1=pdist(d0);

end

function ftree = createfinalpath(ver,qt,vc,qg )
%create final path
qt1=cell2mat(qt(vc+1));
%fedx1 = zeros(length(qt1)+1,2);
%fedx1 = zeros(length(qt1)+1,2);

    for j=1:length(qt1)
        qt2(j,:) = [ver(qt1(j),1),ver(qt1(j),2)];

    end
    qt2(length(qt2)+1,:) = qg;

    ftree = qt2;

end

function co = collision(
x1box,y1box,x2box,y2box,qnear,nearest )
%collision check
% create temporary edge between nearest point in vertecies
and new point
tempedgex = [nearest(1),qnear(1)];
tempedgey = [nearest(2),qnear(2)];

```

```

%plot(tempedgex',tempedgey');
%check intersection point between edges of the obstacle and
new point to
%nearest point edge

[x1i,y1i] = polyxpoly(tempedgex,tempedgey,x1box,y1box);
[x2i,y2i] = polyxpoly(tempedgex,tempedgey,x2box,y2box);

csi1 = size([x1i,y1i]);
csi2 = size ([x2i,y2i]);
%a=csi1(1);
%b=csi2(1);
if (csi1(1)==0 && csi2(1) ==0)
    co = 0;
else
    co= 1;
end

end

function qtreeout = addtreenode( vc,qt,in )
% add new nodes in TREE with minimum cost
qt1=cell2mat(qt(in));

qt1(length(qt1)+1)=vc+1;

qtreeout=num2cell(qt1,[1 2]);
end

```

## **VITA**

Parth Joshi receives B.E Electronics & Communications degree from Gujarat Technological University, Gujarat, India in 2015. He is pursuing his M.S. Electrical Engineering from Texas A&M University, Kingsville, Texas, USA. His area of interest are Embedded systems and Autonomous Vehicles.