Optimistic Motion Planning Using Recursive Sub-Sampling: A New Approach to Sampling-Based Motion Planning

Lhilo Kenye[1,2], Rahul Kala[1] *

[1] Centre of Intelligent Robotics, Indian Institute of Information Technology Allahabad, Jhalwa, Allahabad (India)
[2] NavAjna Technologies Private Limited, HITEC City, Hyderabad (India)

Received 19 July 2020 | Accepted 20 April 2021 | Published 1 April 2022

unir
LA UNIVERSIDAD EN INTERNET

Abstract

Sampling-based motion planning in the field of robot motion planning has provided an effective approach to finding path for even high dimensional configuration space and with the motivation from the concepts of sampling based-motion planners, this paper presents a new sampling-based planning strategy called Optimistic Motion Planning using Recursive Sub-Sampling (OMPRSS), for finding a path from a source to a destination sanguinely without having to construct a roadmap or a tree. The random sample points are generated recursively and connected by straight lines. Generating sample points is limited to a range and edge connectivity is prioritized based on their distances from the line connecting through the parent samples with the intention to shorten the path. The planner is analysed and compared with some sampling strategies of probabilistic roadmap method (PRM) and the experimental results show agile planning with early convergence.

Keywords

Probabilistic Roadmap, Sampling-Based Motion Planning, Robot Motion Planning, Robotics.

DOI: 10.9781/ijimai.2022.04.001

I. Introduction

MOTION planning in robotics is one of the major tasks for any robot and motion planning in itself defines the process of planning how a robot should move from one position (source) to the other (goal). The problem of motion planning is extensively large, which is not just about moving a robot or a part of it from a source to a goal randomly using any path, rather it implies to problems of considering and solving the overall scenario for efficiency and producing effective results of where and how to move even in higher dimensional spaces.

There are generally two types of motion planning: deliberative planning and reactive planning. In deliberative motion planning, the workspace (map) is defined where the robot is given the information about the environment which may include the obstacles, objects to work with, free working area which then can be converted to its configuration space for planning. Whereas in reactive planning, the robot is not aware of what the global environment is like. Here as the robot moves along the environment, it performs actions without pre-planning and is generally used in unpredictable environments. The robot senses and acts in reactive whereas, in deliberative, it senses, plans then acts. In this paper, we introduce a new motion planning algorithm which falls under a type of deliberative motion planning known as sampling-based motion planning. The application of motion planning varies over a wide area which includes Industrial robot arms, planetary exploration, robotic surgery, animation, transportation systems, etc A part from robotics, the sampling-based motion planners are also used in computational biology and animations [1].

* Corresponding author.
E-mail address: lkenye02@gmail.com (L. Kenye), rkala001@gmail.com (R. Kala).

A. Sampling-Based Motion Planners

In sampling-based motion planning the workspace is converted into its configuration space, C. The pose of a robot is described by the configuration of the robot and the set of all configurations is the configuration space. In the configuration space where the source and the goal locations are given, random samples are generated and these samples are generally checked for two conditions: collision-prone (C_{obs}) or collision-free (C_{free}) and the collision-free is considered for connecting the edges. Taking the random C_{free} samples a roadmap or a tree can be introduced for the robot to move through C. Only after a collision free roadmap or path is generated the robot should move around the environment [2]. Sampling based motion planning produces fast solutions in complex maps but may not always give a solution which is why it usually is termed probabilistic completeness [3]. Sampling-based motion planning algorithms or rather planners are classified into roadmap-based planners like the probabilistic roadmap method (PRM) [2] and tree-based algorithms planners like the rapidly exploring random tree (RRT) [4], [5] and in this paper, the new planner introduced is experimentally compared with some variants of PRM. The concepts of sampling-based motion planners are also extended to reactive planning [6], [7].

B. Probabilistic Roadmap Method

The probabilistic roadmap method (PRM) is a roadmap-based planner which is one of the most commonly used sampling-based

motion planners. In a generic PRM planner, for solving in static environments, a set of random sample points are generated, say q_{rand} and from this set the C_{free} samples are considered, while the C_{obs} sample points are rejected. Using $q_{rand} \in C_{free}$ a roadmap is constructed by connecting the samples by C_{free} edges [3], [8]. After a roadmap is constructed, the condition for connectivity is checked, that is, if the source and goal are able to connect to at least one node or vertex of the roadmap without collision. As there can be multiple possible paths from source to goal through the roadmap, this kind of planners are also termed as multi-query planners. Once condition for connectivity is satisfied, graph traversal algorithms like breath-first search [9], Dijkstra, A^*, D^*, anytime A^*, anytime dynamic A^* (AD^*) [3, 10], etcetera, can be used to find the path from source to goal.

C. Quiddity of Optimistic Motion Planning Using Recursive Sub-Sampling

With the motivation from the concepts of sampling-based motion planners, we introduce a new sampling based motion planner, named optimistic motion planning using recursive sub-sampling (OMPRSS), where the planner tries to find the path optimistically without having to construct a roadmap or a tree and the least connectivity condition needs to be checked.

Whenever the straight-line path from source to destination, say L, is collision prone, OMPRSS generates a set of sample points, sort them based on their distances from the L, then tries to find a path through the sorted points. In other words, sorting heuristically prioritizes the points with easiest and shortest points solved first. If the planner fails to find a path through the first set of sample points, a new set of the same number of sample points is generated and the planner attempts to find a path with the help of the previous set of sample points. Every sample point divides the problem into a smaller problem, and the problem is solved recursively. The removal for the need of building a roadmap or a tree helps the planner to find a path faster. From the application point of view, OMPRSS could be used in systems where the time matters and the path length may not be of much concern, for example industrial robots.

The rest of the paper is arranged as follows: literature review is presented in section II, section III discusses the methodology behind the proposed approach, the experiments and results are presented in section IV, section V confers the results and the conclusions in section VI.

II. Literature Review

The geometric sampling-based motion planners are categorized into multi-query based planners which include probabilistic roadmap method (PRM), Lazy PRM, PRM*, Lazy PRM*, SParse roadmap spanner algorithm (SPARS), SPARS2, etcetera, and single-query based planners which includes rapidly-exploring random tree (RRT), RRTConnect, RRT*, Lower bound tree RRT (LBTRRT), Sparse stable RRT, Lazy RRT, bidirectional RRT (BI-RRT) and so on [11]. All the planners have a different approach to sampling and planning strategies and are capable of finding solutions in high dimensional environments. The single-query based planners intend to plan out a collision free path by incrementally building a tree towards the destination either controlled or randomly, whereas multi-query based planners generally build a roadmap and tries to find a path from source to goal through the roadmap.

Kavraki and Latombe [2] presented the elementary working aspect of PRM where the planning approach is deliberative. Here, the planning system first creates a configuration space followed by random sampling and roadmap generation belonging to a collision free field and the robot is intended to move through the constructed roadmap. Different types of PRM have been developed over time which improved the effectiveness and efficiency of PRM. Certain variants can work better in certain type of scenario. Generally, the fundamental concept of PRM stands whereas the sampling strategy is changed in the variants. Bohlin and Kavraki [12] presented a variant of PRM called Lazy PRM which reduces the overall number of collision-checking by first postulating that the roadmap is collision free followed by collision checking while searching for the shortest possible path. The PRM* falls under the category of asymptotically optimal planners, presented by Karaman and Frazzoli [13] which proves the probabilistic completeness and asymptotic optimality of the algorithm. Unlike PRM, PRM* tries to connect to an established set of neighbouring samples. Dobson and Bekris [14] promulgated the SParse roadmap spanner algorithm (SPARS) and SPARS2 which generates sparse roadmaps instead of having to consider the entire number of edges involved in constructing the roadmap.

Some of the other variants of PRM include the Bridge Test, Gaussian sampling PRM and Obstacle-based PRM where the sampling strategies are refined such that the PRM improves its performance in scenarios such as narrow corridors. In this paper, these variants including the uniform sampling PRM [2] are used to compare with the proposed planner. Hsu et al. [15] presented the Bridge Test PRM which hikes up the samples in narrow corridor areas. It basically takes a sample point, say q, and generates another sample point, say q', in the neighbourhood of q if q is collision prone and q' is also checked if it is collision prone. If they are both collision prone, the mid-point, say m of the line $L(q, q')$ is checked whether it is collision free, C_{free}. If m belongs to C_{free}, m is added to the set of vertices for roadmap construction. Boor et al. [16] presented the Gaussian sampling PRM where the samples are generated at a higher density near the obstacles. The Gaussian sampler reduces the number of samples thus improving the efficiency of the planner. Amato et al. [17] promulgated the Obstacle-Based PRM in 3-dimensional scenario where the samples are generated on or near the obstacle surfaces. If the samples are collision prone, they are moved at unit steps towards collision free space until they are in obstacle free space thus, yielding vertices near obstacles, reducing the need of taking in the samples in larger free areas. Several works have been done to compare the various variants of PRM [18, 19].

Lavalle [4] introduced the rapidly-exploring random tree (RRT). RRT has a pulling effect where the tree is iteratively built by expanding towards the random sample point instead of directly connecting to the sample point. Kuffner and LaValle [20] presented a work similar to RRT called the RRT-Connect which iteratively constructs two trees from both the source and the destination and expanded till the trees meets. Similar to the PRM*, Karaman and Frazzoli [13] presented the RRT* which is asymptotically optimal. These planners can be combined to bring out better performances. Kala [21] presented an approach of using both the RRT and PRM. Here the RRT is used for the initial search from several points simultaneously. The points are then used to create a graph through which the path planning is carried out using PRM. The results show significant improvements in the performance. Jason et al. [22] presented a different approach to sampling-based motion planning which, instead of relying entirely on a random sampler, the path planning is enforced by a deterministic approach. Solovey and Kleinbort [23] presented a percolation approach over PRM and its variants leveraging the finesse of the variants. Ichter et al. [24] presented an approach which learns to identify essential regions or samples (for example, a doorway) which is enforced by graph theory and neural networks. Vonásek et al. [25] presented a sampling-based motion planning method which first finds approximate solutions using scaled-down robot and uses the approximated solutions as a guide to find the actual path between the source and the goal. Kim et al. [26] presented lazy collision checking approach which adaptively checks collision region by generating an approximating free configuration space allowing the system to obtain clues on which region should be checked first.

The sampling-based planners have been extensively used in several works. Švestka and Overmars [27] proposed a strategy of collaborative and coordinated motion planning for multiple robots using sampling-based planners. Clark [28] also presented a work on multi-robot motion planning using PRM. Yao and Gupta [29] also presented the use of sampling-based motion planners for end-effector path planning. Sampling based motion planners are also embedded in Open Motion Planning Library (OMPL) [11] which is being extensively used. The sampling-based motion planners has led to an amplification of new problem-solving domains. Kala [30] presented a work on using multiple robots for performing mission driven tasks with the help of sampling-based motion s planners and its concepts.

III. Optimistic Motion Planning Using Recursive Sub-Sampling

Optimistic motion planning using recursive sub-sampling (OMPRSS) is a new approach to path finding based on the concept of sampling-based motion planning. The approach is considered to be optimistic as the algorithm always assumes that there is a collision free path between the source and the goal or the sub-source or sub-goal, where the source and the goal are always connected by straight lines. Only when there is a collision prone path, the algorithm generates new set of controlled random points in an attempt to find the path through the sub-source and sub-goal. OMPRSS is exclusive of trees and roadmaps. Given a map with the source and the goal, the planner first connects the source s and the goal g with a straight line, say $L(s, g)$. Then this line is checked for collision. If the line is collision free, say C_{free}, it is taken as the path as shown in Fig. 1, which is the shortest possible path from s to g. An assumption here is $C_{free} = R^N$.

In the following discussions, the terms C_{free} and C_{obs} indicates collision free and collision prone respectively, while C_{free} and C_{obs} represent the collision free configuration space and collision prone configuration space respectively. Also, a line L with at least a point in C_{obs} is considered a C_{obs} line.

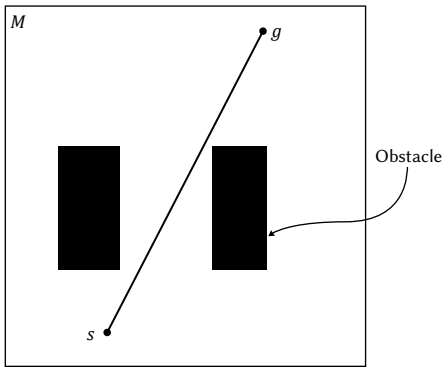

Fig. 1. A straight-line collision free path from source to goal. M denotes the map.

If the line $L(s, g)$ is collision prone, a set of uniform random sample points $q \in Q \subset C_{free}$ are generated, which are collision free with the hope that the path from s to g will be collision free. A certain range, say R, is set for generating the random points. This range factor is maintained to limit the area of generating the random points with the intention to get a shorter path length. The range factor is defined as the diagonal distance of the space aligned to the coordinate axis system, within which the sample points q are generated. As an example, consider the heuristic that the robot never travels such that the projection of motion on the straight-line path is negative. Now the sampling region can be limited to the hyperspace bounded by s and g, space aligned along the coordinate axis system with the range factor as $d(s, g)$ where d is the distance function. The range factor could start from the length of $d(s, g)$ to the upper and lower limit of the map, i.e., the entire configuration space. Fig. 2 shows a condition where $d(s, g)$ is collision prone, C_{obs}, and the range of generating uniform random sample points is within the n-orthotype or hyper rectangular area with the length of $d(s, g)$ as its diagonal. Considering the coordinate points of s as $(s_1, s_2, s_3, ..., s_n)$ and g as $(g_1, g_2, g_3, ..., g_n)$, the area would be: for one corner of the hyper rectangle the point will be $c_1 = [min(s_1, g_1), min(s_2, g_2), ..., min(s_n, g_n)]^T$ and the other corner as $c_2 = [max(s_1, g_1), max(s_2, g_2), ..., max(s_n, g_n)]^T$.

Consider the case given in Fig. 2. Since $L(s, g)$ is a C_{obs} line, we generate a set of random points, say four points Q_r: $\{q_1, q_2, q_3, q_4\}$ within a rectangular area – area resulting from the range factor taken as length of $d(s, g)$. The generated random sample points are sorted according to the distance of each point to the line $L(s, g)$. The farther a point is from the line, the lesser the priority of that point in the population of random sample points. The notion is to sample a point q such that s to g via q is a path. Closer be a point to the line from s to g, more likely it is to produce a shorter path. Moreover, collision checking effort is proportional to the length of the path. Closer be a point to the line, lesser is the computational cost. Let us consider that, after sorting according to the distance, the points are arranged as Q_p: $\{q_2, q_4, q_3, q_1\}$. Here, Q_p is a priority queue of the points where for further task the planner first chooses the point with the highest priority (least distance from q_i to $d(s, g)$), in this case point q_2. Before going further, let us first look into the method of prioritizing the sample points.

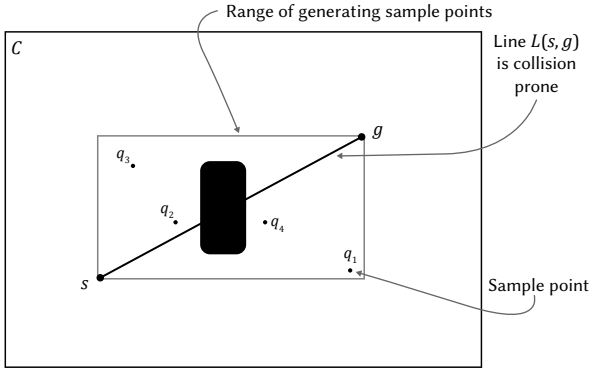

Fig. 2. Range set to length of $L(s, g)$ resulting in a rectangular area for generating sample points. C denotes the configuration space.

A. Sample Prioritization

As mentioned, the random points belonging to C_{free} can be prioritized based on the distance of a point q_i to the line $L(s, g)$ All the sample points generated randomly will be at a particular distance from the line $L(s, g)$. Finding the distances of each point from $L(s, g)$ can help prioritize them. One way of determining the point distance to a line is measuring the normal distance, which basically is the shortest distance of a point to a line.

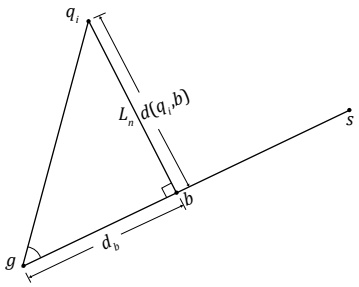

Fig. 3. An illustration of the normal distance of point q_i to line $L(s, g)$, $d(q_i, b)$ which would be used for prioritizing the sample point.

In Fig. 3, points $s = (x_s, y_s)$ and $g = (x_g, y_g)$ are the location of source and goal respectively. The point $q_i = (x_i, y_i)$ is the location of a random sample point generated and the point $b = (x_b, y_b)$ is the intercept of the normal line, say L_n, to $L(s,g)$. Here, we are interested in determining the distance of point q_i to the line $L(s,g)$, say $d(q_i, b)$ as depicted in Fig. 3. Let the distance between point g and b be d_b. We can deduce that

$$(q_i - g).(s - g) = |q_i - g|\,|s - g|\,cos\theta \tag{1}$$

The line L_n is normal to $L(s,g)$ forming a right triangle Δgbq_i. We have:

$$d_b = |q_i - g|\,cos\theta \tag{2}$$

$$\Rightarrow cos\theta = \frac{d_b}{|q_i - g|} \tag{3}$$

Replacing the expression of $cos\theta$ in Eq. (1), we have:

$$(q_i - g).(s - g) = |q_i - g|\,|s - g|\,\frac{d_b}{|q_i - g|} \tag{4}$$

$$d_b = \frac{(q_i - g).(s - g)}{|s - g|} \tag{5}$$

$$d_b = (q_i - g).\hat{u}(s - g) \tag{6}$$

Where, $\hat{u}$ is a unit vector. We can determine $d(q_i, b)$, depicted in Fig. 3 as:

$$d(q_i, b) = \sqrt{|q_i - g|^2 - d_b^2} \tag{7}$$

The sample points can be prioritized based on the distances of each point from the line $L(s,g)$ which can be obtained using Eq. (7).

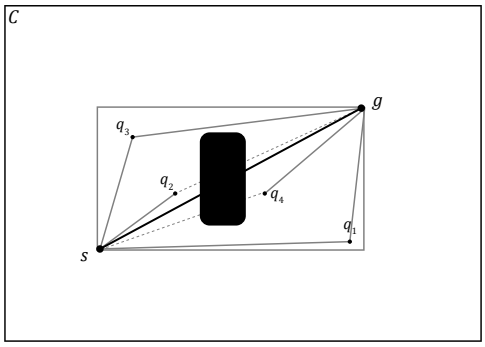

Fig. 4. Path from s to g through the sample points (q_1, q_2, q_3, q_4) generated within a specified range or area. Dotted lines denote collision-prone paths.

B. Working Principle of OMPRSS

In this section, we will discuss the working of the planner when the straight-line path from source to goal is collision prone. If $L(s,g)$ is collision prone we generate a set of uniform random points $Q \subset C_{free}$. These points are prioritized using the approach of point prioritization discussed in the previous section. Considering the case given in Fig. 2, we generated four C_{free} random points and prioritized them as Q_p: {q_2, q_4 q_3, q_1}. The purpose of prioritization is to go about with the point with the highest priority for the next step.

After the points are prioritized, a path from s to g is checked through each sample points, shown in Fig. 4. As point q_2 has the higher priority in this case, we go about with q_2 by first checking whether the straight-line path from s to q_2 either belongs to C_{free} or C_{obs} and followed by checking the straight line path from q_2 to g. If a C_{free} path is found from s to g through $L(s, q_2)$ and $L(q_2, g)$, the planner returns with a path connecting the source to goal through the sample point q_2. But as in the case of Fig. 4, since $L(q_2, g)$ is collision prone (C_{obs}), we consider the next point in the priority queue q_4, which is the next closest point to $L(s, g)$. The path from s to g is still C_{obs} as the line $L(s, q_4)$ is C_{obs}. Next, we consider q_3. Here, both $L(s, q_3)$ and $L(q_3, G)$ are C_{free}, so in this case the planner returns a successful path. The point q_1 also produces a C_{free} path, but as its priority is lesser compared to q_3, the path through q_1 is ignored. Through the example of Fig. 4 we can conclude that point prioritization helps in finding a shorter path as any point closer to the line would produce a shorter path length than that of a point farther to the line of concern.

Let us consider a different case shown in Fig. 5. In this case, we can observe that all sample points produce C_{obs} path. This planner works recursively, where in this type of cases we repeat the same procedure as we did for the case of Fig. 4 with the sample point as the new source or goal depending on which of the line ($L(s, q_i)$ or $L(q_i, g)$) is C_{obs}. In Fig. 5, we can observe two cases:

- Firstly, one of $L(s, q_i)$ or $L(q_i, g)$ is C_{obs} as in cases of points q_1, q_2 and q_4. Here, in case of points q_1 and q_2 the sample points become the new goals and the source remains the same in the next recursion. Whereas in case the of q_4, the goal remains the same as it is and the sample point becomes the new source.
- The second case is when both $L(s, q_i)$ and $L(q_i, g)$ are C_{obs}. In this case, we change the priority of that point by multiplying with a penalty. The value of the penalty can be chosen based on the distances produced by all sample points such that after multiplying with the penalty, the distance of that point increases so that the priority of that point gets lesser than the point with the largest distance but only has one C_{obs} line initially. If all the paths produced the random points are C_{obs}, a penalty is multiplied to all points and in this condition, all points get back to its original priority list. This is done to reduce computational complexities as for a point with both lines to be C_{obs}, the number of recursions would increase. If we first work on points with only one C_{obs} line and find a path, it would reduce the depth of recursion as compared to points having both lines to be C_{obs}. When, for all sample points, if both $L(s, q_i)$ and $L(q_i, g)$ are C_{obs}, we can avoid the random points and generate new points or start the planner all over, but if we keep avoiding points when all points produce both paths as C_{obs}, the time cost could gradually increase. In Fig. 5, point q_3 has both $L(s, q_3)$ and $L(q_3, g)$ as C_{obs}. The priority queue changes from Q_p: {q_1, q_2, q_3, q_4} to Q_p: {q_1, q_2, q_4, q_3}.

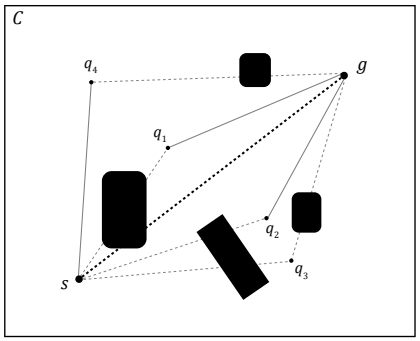

Fig. 5. All sample points producing collision-prone (C_{obs}) paths.

Summarizing, the collision aware priority is given by:

$$\pi' = d\perp(q_i, L)(1 + \alpha C_1 + \beta C_2) \tag{8}$$

Where, α and β are very large constants and C_1 and C_2 can be defined as

$$C_1 = \begin{cases} 0, & if\ L(s, q_i)\ is\ C_{free} \\ large\ number, & otherwise \end{cases} \tag{9}$$

$$C_2 = \begin{cases} 0, & if\ L(q_i, g)\ is\ C_{free} \\ large\ number, & otherwise \end{cases} \quad (10)$$

When a line from source to sample point or sample point to the goal (or both) is C_{obs}, we get into a recursion where the same set or number of C_{free} random points are generated. Let us consider the point q_1 in Fig. 5. The path from q_1 to g is C_{free} but the path from s to q_1 is C_{obs}. As a result, we generate another four random points, say Q_{11}: $\{q_{11}, q_{12}, q_{13}, q_{14}\} \in C_{free}$ within the range, say between s and q_1. The area would be a square or rectangle with $L(s, q_1)$ as the diagonal. The same procedure is followed with the intention to find a path from s to q_1 by checking for C_{free} paths through one of the new sample points with q_1 as the new goal. Once a path is found, the original source and goal can get connected via the sample points as shown in Fig. 6. In this case, the path would be $s \rightarrow q_{13} \rightarrow q_1 \rightarrow g$. If a path is not found, from s to g through the sample points Q_{11}: $\{q_{11}, q_{12}, q_{13}, q_{14}\}$, we continue with the same procedure using Q_{11} as the new source or goal, resulting into another depth of recursion. The limit of recursion can be set to a certain depth. Change in the depth limit can also change the response of the planner. The procedure of generating the random points is repeated until a path is found or the depth limit is reached. We can observe that by maintaining a range the total path length can be reduced but, setting the range factor has some issues which we will discuss in the next section.

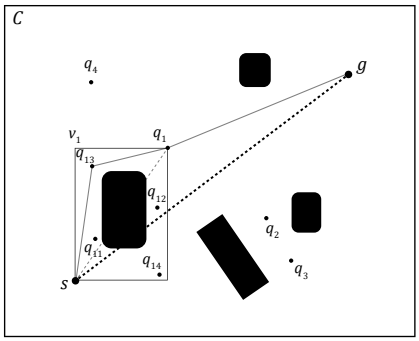

Fig. 6. Illustration of new sample points (q_{11}, q_{12}, q_{13}, q_{14}) generated in recursion of depth 1. V_1 denotes the new area of generating the sample points setting the range factor to be $L(S, q_1)$.

C. Repercussion of Range and Proposed Solution

We have observed that setting a range factor can greatly reduce the path length, but there are some issues which pops up with it. In this section, we will discuss some of the possible situations where the range factor becomes an issue and the solutions to it. Let us first consider the issues with regard to the range factor set to the length $L(s, g)$, say d_L. To understand these issues, let us take a 2D scenario as an example. There are two possible cases where the planner can get into a no-solution state:

- When $L(s, g)$ becomes perpendicular to x-axis: Here, the slope of $L(s, g)$ becomes infinite.

$$m = \frac{\Delta y}{\Delta x} = \frac{y_g - y_s}{x_g - x_s} \quad (11)$$

Here:

$$x_g = x_s \Rightarrow m = \frac{y_g - y_s}{0} = undefined \quad (12)$$

- When $L(s,g)$ becomes parallel to x-axis: The slope of the normal to $L(s, g)$ will be undefined as change in y will be zero.

$$m = \frac{0}{x_g - x_s} = 0 \quad (13)$$

Here, slope of normal to the line $L(s, g)$ will be

$$s = -\frac{1}{m} = undefined \quad (14)$$

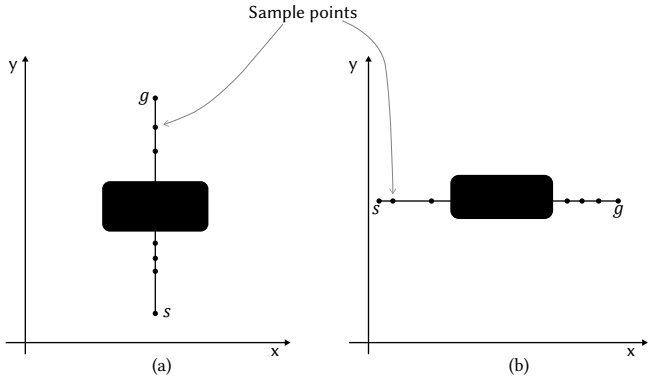

Fig. 7. Representation of the two cases where the range for defining the area of generating random sample points would fail to produce points to find a path. (a) The line $L(s, g)$ perpendicular to the x-axis, (b) The line $L(s, g)$ parallel to x-axis. In both cases, the points are generated along the line $L(s, g)$ for choosing range factor as d_L.

Finding the distance of a point to the line will not be an issue in both cases as the distance in the case of the first problem will be the difference in the x values of the sample point and the intercept of the line and for the second, the difference in y values. The problem is that the volume of generating random sample points will be the line itself, i.e., the points will lie on the line as shown in Fig. 7. In these cases, no path can be found.

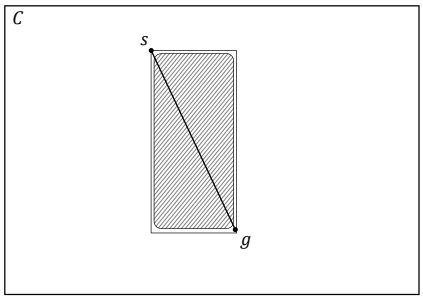

Fig. 8. An illustration where OMPRSS has high possibility of failing due to less C_{free} region or area for generating sample points. The inner most rectangular shape filled with lines indicates an obstacle, say O. s and g are the source and goal respectively and the rectangular box, say r with the diagonal line connecting s and g is the area for generating sample points. The space between the O and r is the C_{free} region for generating the sample points.

Another problem that can occur is the situation given in Fig. 8. In this case, the source and the goal are both too close to the obstacle which results in small C_{free} area to generate the C_{free} points. The planner in this case has very thin chances to find a path from source to goal because the points can get too close to each other with the increase in the depth of recursion and may even reach to an extend where the points are generated in the same point in the configuration space resulting in generating zero distances. A similar case would be as illustrated in Fig. 10 where, if the range factor is taken as d_L, there is no way to find a path. All points generated will not be able to find out a path as long as the range factor is d_L in this case.

The solution to the above problems would be increasing the range of generating the points by some factor which will eventually increase the volume of sample point generation. Various range factors can be assigned to the planner such that the prospect of finding a path is

increased. The degree to which the range factor increases is flexible and there are no discrete methods to increasing it. In this paper, we propose a particular way of increasing the volume. Instead of using d_L, which is the length of $L(s, g)$, for setting up the volume of generating the sample points, the line $L(s, g)$ is extended by an extension factor, say η, such that the volume increases. Here, both the ends of $L(s, g)$ are extended by η. In other words, both s and g are either added or subtracted by η depending on where s and g are located. If the location of source is $s = (s_1, s_2, ..., s_n)^T$ and goal is $g = (g_1, g_2, ..., g_n)^T$, the expression for volume extension can be written as:

$$s' = \begin{cases} s_i + \eta, & if\ (s_i - g_i) \geq 0, \forall i \in n \\ s_i - \eta, & if\ (s_i - g_i) < 0, \forall i \in n \end{cases} \quad (15)$$

$$g' = \begin{cases} g_i - \eta, & if\ (s_i - g_i) \geq 0, \forall i \in n \\ g_i + \eta, & if\ (s_i - g_i) < 0, \forall i \in n \end{cases} \quad (16)$$

Here, s' and g' are the new corner points of the volume and $L(s', g')$ becomes the diagonal line of the extended volume. In both Eq. (15) and (16), the greater or equal condition ($\geq$) is used in order to deal with cases when $L(S, G)$ is perpendicular or parallel to one of the axes. The case where $L(S, G)$ is perpendicular or parallel to any axes occurs when for $i = 1, 2, ..., n$, $(S_i - G_i) = 0$.

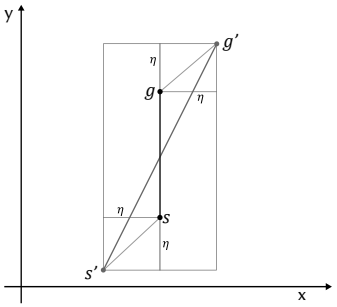

Fig. 9. Illustration of volume extension for generating the sample points by η for $L(s, g)$ perpendicular to the x-axis. The extension operation redefines the volume such that $L(s', g')$ is the diagonal of the incremented volume.

One of the possible extension factors could be the half of the length of $L(s, g)$, that is $\eta = 0.5 \times d_L$. Let us once again consider a 2D scenario (Fig. 9) with source as (x_s, y_s) and goal as (x_g, y_g) where, $x_g > x_s$ and $y_g > y_s$. Let the length of $L(s, g)$ be d_L. Based on the extension condition, the end points after extension of d_L would be

$$(x_s', y_s')^T = (x_s - \eta, y_s - \eta)^T \quad (17)$$

$$(x_g', y_g')^T = (x_g + \eta, y_g + \eta)^T \quad (18)$$

This will result in a larger volume in the configuration space for generating the random points as shown in Fig. 10. The factor can even be increased if required such as using the whole d_L as the factor, that is $\eta = d_L$ or even up to the extent of using the whole configuration space as the range. The chances of getting a path may increase in η, but as it increases, the chances of getting points farther away from the line are more, as a result we might end up with a larger path length.

If the range factor is changed, the process of point prioritization needs to be changed to some extend as problems could arise in condition where the point is beyond the actual line as shown in Fig. 11. Here, a normal line does not pass through the line $L(s, g)$ as a result the point of intercept B would go beyond $L(s, g)$. This issue is taken into consideration because though a point may lie the closest to the line based on the normal line it may still be quite far away from the source and the goal, which when given with higher priority may result in a longer path than those points which have larger normal distance. This problem can be solved by comparison of the distance between $d(s, g)$, $d(s, b)$ and $d(b, g)$. First the following condition as given in Eq. (19) needs to be checked:

$$d\perp(q_i L) = \begin{cases} d(q_i, b), if\ d(s, g) = d(s, b) + d(b, g) \\ \min\left(d(q_i, s), d(q_i, g)\right), \quad otherwise \end{cases} \quad (19)$$

We can conclude from these issues that the larger the range factor, the chances of getting a shorter path reduces. The range factor can be configured based on the type of map the robot is dealing with.

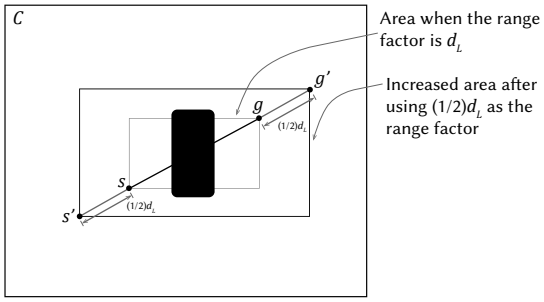

Fig. 10. A representation of using a factor of half of d_L to increase the volume for generating random points. Here, d_L is the length of $L(s, g)$.

D. Pseudo Code

The pseudo code for OMPRSS is given in Algorithm 1. The function $L(s, g)$ mentioned in Algorithm 1 is given in Algorithm 2. The function *connect* $(s, g, depth)$ returns two values. First is the connectivity *con* and second is the path τ. If the source and the goal is connected via a path, a true value is returned and all the points connecting the source and the goal (including the source and the goal) are returned in τ. The function *distanceToLine*(q_i, s, g) returns the distance of a sample point q_i to the line $L(s, g)$. Here, s and g will keep changing with every depth as the sample point will become new source or goal with every recursion depth. The distances are used for sorting which actually is the step of prioritizing. The parameter *limit* in Algorithm 1 is the limit or threshold for the recursion depth.

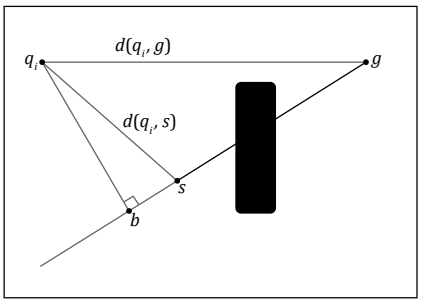

Fig. 11. An illustration of the intercept b laying beyond the line $L(s, g)$ due to volume extension.

IV. Experiments and Results

The experiment was conducted on two dimensional maps, bitmap images, considering the obstacles to be 0s and the collision free area to be 1s. The robot is taken to be a point robot inferring that the workspace need not be converted to its configuration space. The results on the working of OMPRSS is first presented followed by the response of the planner to different parameter alterations and comparison with other sampling-based motion planners.

Algorithm 1: Optimistic motion planner using recursive sub-sampling

$[con, \tau] = connect(s, g, depth)$

Input: source (s), goal (g) and current recursion depth ($depth$)

Output: $con = true$ if connected, $false$ if no path found, τ = Path from s to g

1. if $L(s, g) = true$ then
2. $con = true, \tau = [s, g]$
3. $return\ [con, \tau]$
4. end if
5. if $depth > limit$ then
6. $con = false$
7. return $[con, NIL]$
8. end if
9. $\tau = \phi$
10. if $L(s, g) = false$, Q = uniform random points, $n = 1, 2, \ldots$ then
11. for $i = q_i \in Q$ do
12. $\pi(q_i) = distanceToLine(q_i, s, g)$
13. if $L(s, q_i) = false$ or $L(q_i, g) = false$ then
14. $\pi'(q_i) = \pi(q_i)(1 + \alpha C_1 + \beta C_2)$ // Eq. (8)
15. end if
16. end for
17. Sort Q based on $\pi'(q_i)$
18. for $i = 1$ to n do
19. if $L(s, q_i) = false$ and $L(q_i, g) = true$ then
20. $[con, \tau] = connect(s, \pi'(q_i), depth + 1)$
21. end if
22. if $L(s, q_i) = true$ and $L(q_i, g) = false$ then
23. $[con, \tau] = connect(\pi'(q_i), g, depth + 1)$
24. end if
25. if $L(s, q_i) = false$ and $L(q_i, g) = false$ then
26. $[con_1, \tau_1] = connect(s, \pi'(q_i), depth + 1)$
27. $[con_2, \tau_2] = connect(\pi'(q_i), g, depth + 1)$
28. $con = (con_1 \wedge con_2)$
29. $\tau = [\tau_1, \tau_2]$
30. end if
31. if $con = true$ and $NotEmpty(\tau)$ then
32. return $[con, \tau]$
33. end if
34. end for
35. $con = false$
36. return $[con, \tau]$
37. end if

Algorithm 2: Line-Collision check

$L(S, G)$

Input: Points source (s) and goal (g)

Output: true if straight line from s to g is C_{free}, else $false$

1. $s = source, g = goal$
2. for $i = 0$ to 1 in small steps do
3. if $i \times s + (1-i) \times g \in C_{obs}$ then
4. end if
5. end if
6. end for
7. return $true$

A. Results on the Working of OMPRSS

The initial experiment was performed on the basic working method of OMPRSS. The maps used were 500 ×500 bitmap images. The results are shown in Fig. 12(a) with the source at (50, 50) and the goal at (100, 450) and Fig. 12(b) with the source at (50, 50) and the goal at (350, 300), both had four sample points generated per recursion and the recursion depth limit was kept at 4. The range factor used was set to half of d_L. The time required to find the path in the case of Fig. 12(a) was 0.05681274 seconds and the cost (path length) was 690.2206, while 0.8073688 seconds and 453.9041 in case of Fig. 12(b). In the figures, the entire generated sample points are not indicated besides the sample points resulting to the path formation. As the sample points are generated randomly, different path lengths can be generated each time the planner is executed. Experimental results are shown in Fig. 13.

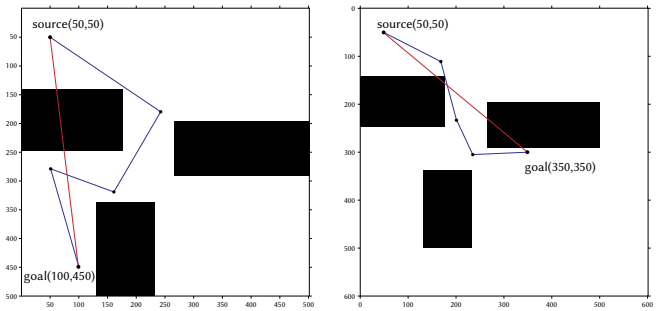

Fig. 12. OMPRSS path planning results in a bitmap image of size 500 ×500. (a) and (b) illustrates the paths found by OMPRSS in different maps where, in both cases, the planner generates sample points in C_{free} region to connect the source and goal as the straight lines connecting them are collision prone.

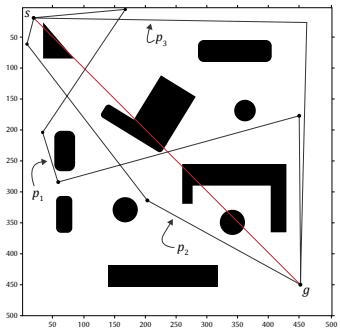

Fig. 13. Response of OMPRSS at different executions over the same source and goal locations - source at (10,10) and goal at (450,450), using the same range. The range used was half of d_L The path p_1 with length 1154, path p_2 with length 866 and p_3 with path length 645.

B. Response to Changes in Parameters

There are a few parameters which can be changed in OMPRSS. They include:

- Range (which determines the area of generating random sample points)
- Sample points
- Depth of recursion

1. Altering the Range Factor

For generating the random sample points three variations of range (variants discussed in section III.B) were used for which all the variants responded differently in different scenarios. The variants of range are:

- Half the line of $L(s, g)$, i.e. $0.5 \times d_L$. Let us consider this range as R_1

- The length of $L(s,g)$, d_L. Let this range be R_2
- The entire map. Let this range be R_3

The maps used for this experiment are given in Fig. 14. Table I shows the observations of different range factors responding to different scenarios. Map size used: 500 ×500. Number of sample points used is 4. Depth (or limit) of recursion is 4. Cost is the path length and the time taken are in seconds.

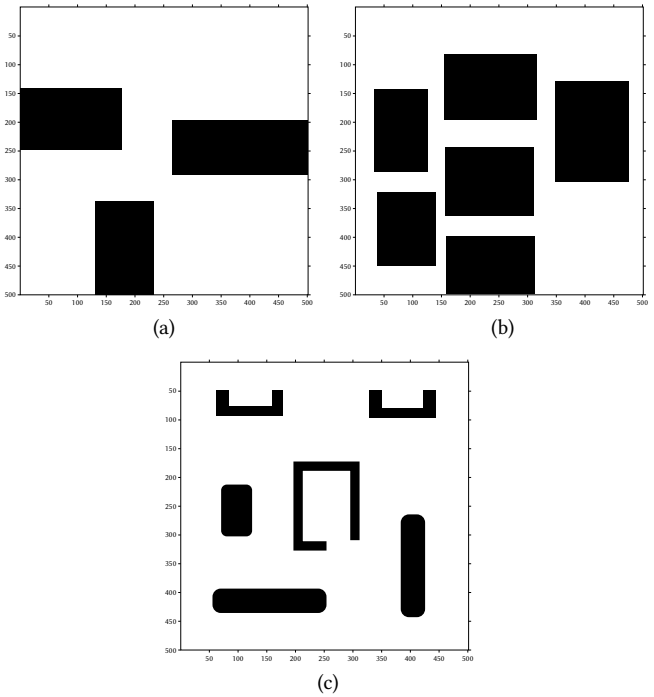

Fig. 14. 500 ×500 bit maps used for comparison of the response of OMPRSS over change in range factors. (a) as M_1, (b) as M_2 and (c) as M_3 in Table I.

2. Altering the Number of Sample Points

The other parameter that can be changed is the number of sample points generated at each recursion. The experiment was performed on map M_3 (Fig. 14(c)). The depth of recursion was 4 and the range factor was half of d_L. Fig. 15 and Fig. 16 shows the results of the experiment. Here the number of sample points is incrementally increased at a step of 1 per iteration from 4 to 53, whereas the map, depth of recursion, range factor and the location of both source and goal are not changed.

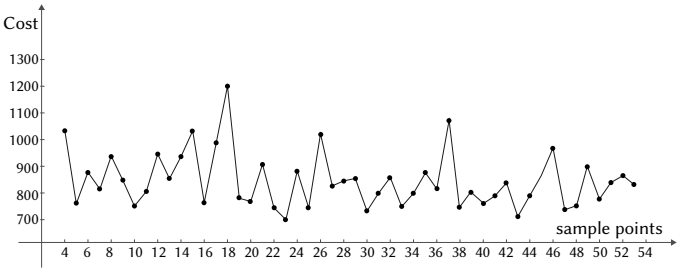

Fig. 15. Plot of cost versus sample points: showing the response of OMPRSS over change in sample points on M_3 (Fig. 14(c)) with source at (10, 10) and goal at (470, 470).

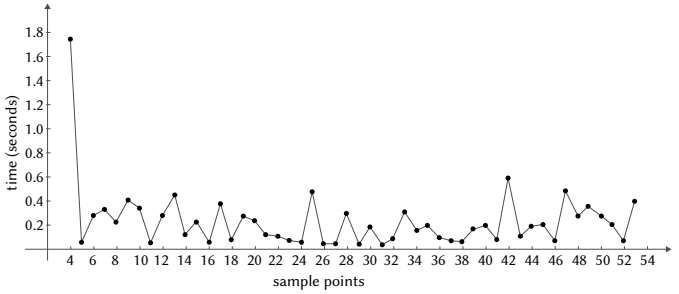

Fig. 16. Plot of time versus sample points: showing the response of OMPRSS over change in sample points on M3 (Fig. 14(c)) with source at (10, 10) and goal at (470, 470).

TABLE I. Comparison of Responses of the Planner Over Different Range Factors in Different Scenarios. The Maps Used Are Given in Fig. 14 As M_1, M_2 and M_3

Sl. No.	Map	Source	Goal	Range	Cost	Time (seconds)	Result
1	M_1	(20,20)	(450,450)	R_1	**802.6841**	0.04598	Pass
2	M_1	(20,20)	(450,450)	R_2	*941.3936*	0.05159	Pass
3	M_1	(20,20)	(450,450)	R_3	1006.8177	0:02251	Pass
4	M_1	(100,100)	(10,300)	R_1	*895.1789*	5.13672	Pass
5	M_1	(100,100)	(10,300)	R_2	**783.3984**	3.51435	Pass
6	M_1	(100,100)	(10,300)	R_3	930.3145	0.01224	Pass
7	M_2	(10,10)	(450,450)	R_1	741.5306	0.76386	Pass
8	M_2	(10,10)	(450,450)	R_2	**727.5929**	0.56532	Pass
9	M_2	(10,10)	(450,450)	R_3	*769.5605*	0.13955	Pass
10	M_2	(50,300)	(450,100)	R_1	**664.7577**	0.76386	Pass
11	M_2	(50,300)	(450,100)	R_2	*708.8577*	0.081249	Pass
12	M_2	(50,300)	(450,100)	R_3	729.6059	0.43471	Pass
13	M_3	(100,70)	(450,470)	R_1	*716.774*	0.062411	Pass
14	M_3	(100,70)	(450,470)	R_2	**709.3188**	0.014065	Pass
15	M_3	(100,70)	(450,470)	R_3	1373.9801	0.073104	Pass
16	M_3	(100,70)	(230,200)	R_1	---	---	Fail
17	M_3	(100,70)	(230,200)	R_2	---	---	Fail
18	M_3	(100,70)	(230,200)	R_3	**754.2889**	1.12045	Pass

3. Altering the Depth of Recursion

Keeping the number of sample points constant including the map, range factor, and the source and goal locations while altering the depth of recursion, OMPRSS responded differently. The experiment was performed over M_3 (Fig. 14(c)). Number of sample points kept constant at 4, source at (10, 10) and goal at (470, 470) and half of d_L as the range factor. The depth of recursion was initiated at 4 and not at 1 because at a depth of 1, the planner is likely to fail. The reason behind starting at 4 is to start off at a likely possible successful planning followed by increasing the depth to observe the response over incrementally increasing the depth. The experimental results are shown graphically in Fig. 17 and Fig. 18.

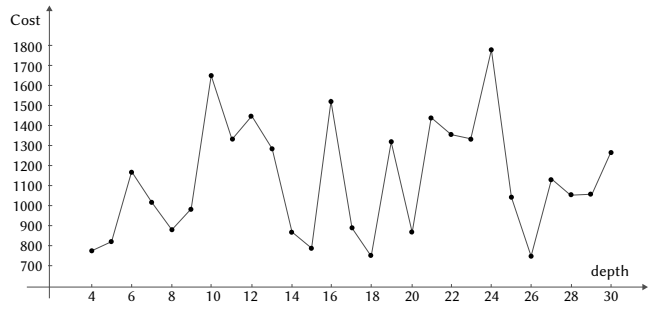

Fig. 17. Plot of cost versus depth: showing the response of OMPRSS over change in depth of recursion on M_3 (Fig. 14(c)) with source at (10, 10) and goal at (470, 470).

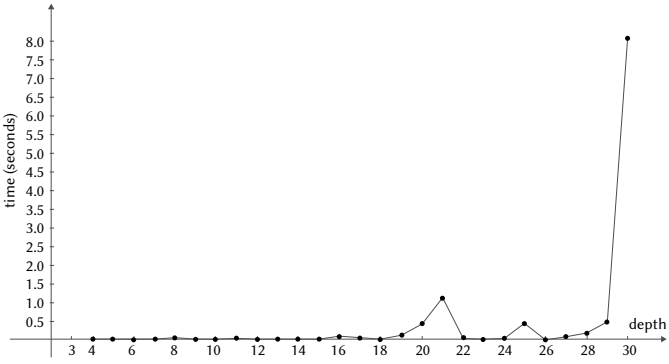

Fig. 18. Plot of time versus depth: showing the response of OMPRSS over change in depth of recursion on M_3 (Fig. 14(c)) with source at (10, 10) and goal at (470, 470).

C. Comparison of OMPRSS With PRM Variants

In this section we will look into the performance of OMPRSS against some of the variants of probabilistic roadmap method (PRM). The PRM variants used are:

- Bridge test PRM
- Gaussian sampling-based PRM
- Obstacle-based PRM
- Uniform cost search PRM

The comparison was made based on three outcomes:

- Cost inferring the path length
- Time required for planning
- Success rate specifying the number of successful planning

The experiment was performed to view the response of both the PRM variants and OMPRSS by increasing the sample points. The first comparison experiment was performed on M_6 from Fig. 19, the location of source at (10, 10) and goal at (450, 450).

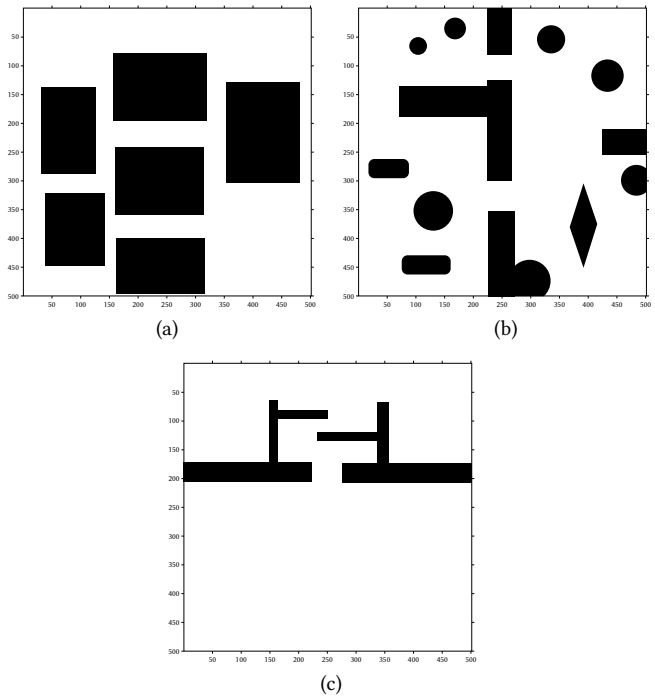

Fig. 19. 500×500 bit maps used for comparison of PRM variants and OMPRSS (a) as M_4, (b) as M_5 and (c) as M_6.

The initial number of sample points for the PRM variants was 200 incremented by 50 per iteration, iterated 80 times resulting to 4150 sample points at the end. Whereas for OMPRSS, the initial number of sample points was 5 incremented by 1 per iteration, iterated 80 times. For both PRM variants and OMPRSS, for a particular number of sample points (that is in one iteration), say S_i, the planner was executed 25 times, and from the generated results, of each S_i, the average of the cost, the execution time and the success rate were considered. For OMPRSS, the range factor used was half of d_L and the depth of recursion was kept at 4. If any planner fails to find a path in any of the 2000 times executed (i.e. 80×25 times), the cost can be taken to be a high value - in the experiment, cost was taken as 9999 if any planner fails. And also, if a planner fails to find any path for the 25 times executed per S_i the success rate would be 0. On the other hand, if a planner succeeds to find paths in all 25 times, the success rate would be 1.

Fig. 20, Fig. 21 and Fig. 22 show the graph for PRM variants of cost, time and success rate versus increasing sample points respectively. Whereas Fig. 23, Fig. 24 and Fig. 25 show the result of OMPRSS for cost, time and success rate against the increasing sample points. In order to view the successful planning and cost convergence against time, the data generated from both PRM variants and OMPRSS were plotted. Fig. 26 and Fig. 27 shows the scatter plot of cost versus time and success rate versus time generated by PRM variants respectively, and Fig. 28 and Fig. 29 for OMPRSS.

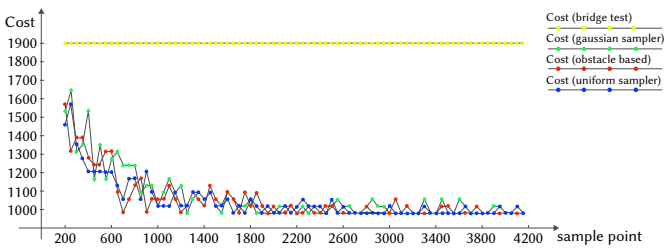

Fig. 20. Graph of cost versus increasing sample points generated by the PRM variants in M_6.

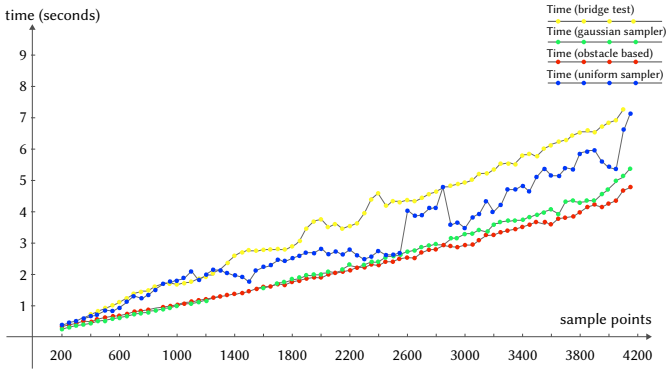

Fig. 21. Graph of time versus increasing sample points generated by the PRM variants in M_6.

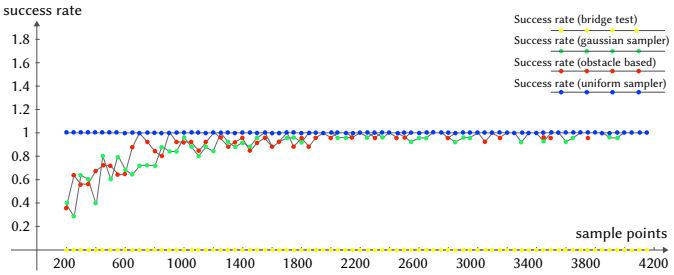

Fig. 22. Graph of success rate versus increasing sample points generated by the PRM variants in M_6.

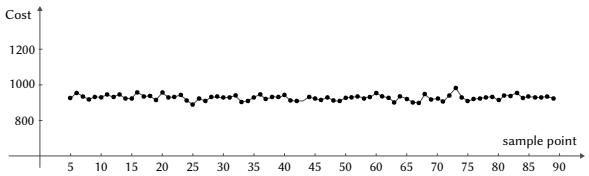

Fig. 23. Graph of cost versus increasing sample points generated by the OMPRSS in M_6.

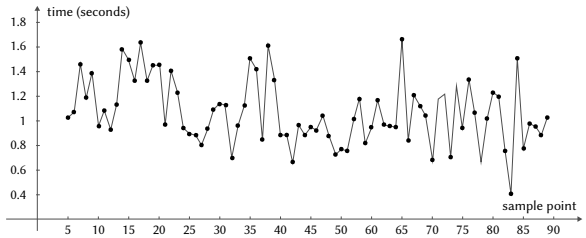

Fig. 24. Graph of time versus increasing sample points generated by the OMPRSS in M_6.

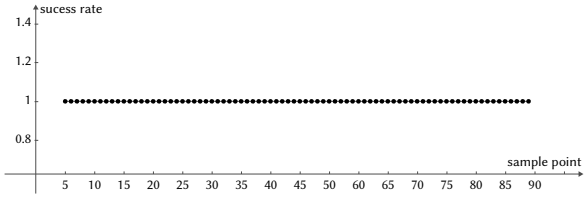

Fig. 25. Graph of success rate versus increasing sample points generated by the OMPRSS in M_6.

TABLE II. Cost Comparison of the PRM Variants and OMPRSS in Different Scenarios (Maps in Fig. 19) at Various Time Steps. Source and Goal Kept at (10, 10) and (450, 450) Respectively for All Maps

Map	Time (sec)	Bridge test	Gaussian sampler	Obstacle-based	Uniform sampler	OMPRSS
M_4	Less than 0.04	9999	9999	9999	9999	9999
M_4	Less than 0.08	9999	9999	9999	9999	**890.5013**
M_4	Less than 0.12	9999	9999	9999	9999	**890.5013**
M_4	Less than 0.16	9999	9999	9999	9999	**890.5013**
M_4	Less than 0.2	9999	9999	9999	9999	**890.5013**
M_4	Less than 0.24	9999	8527.6245	**806.7357**	9999	*890.5013*
M_4	Less than 0.28	9999	7425.3433	**806.7357**	9999	*890.5013*
M_4	Less than 0.32	9999	7425.3433	**806.7357**	9999	*890.5013*
M_4	Less than 0.36	9999	5585.5652	**806.7357**	9999	*890.5013*
M_4	Less than 0.4	9999	5585.5652	*806.7357*	**782.1605**	890.5013
M_5	Less than 0.01	9999	9999	9999	9999	9999
M_5	Less than 0.06	9999	9999	9999	9999	**774.6329**
M_5	Less than 0.11	9999	9999	9999	9999	**774.6329**
M_5	Less than 0.16	9999	9999	9999	9999	**774.6329**
M_5	Less than 0.21	9999	9999	9999	9999	**774.6329**
M_5	Less than 0.26	9999	9999	9999	9999	**774.6329**
M_5	Less than 0.31	9999	3750.8509	*1160.7313*	9999	**774.6329**
M_5	Less than 0.36	9999	1178.5719	*794.7315*	9999	**774.6329**
M_5	Less than 0.41	9999	1166.7064	794.7315	*786.6865*	**774.6329**
M_5	Less than 0.5	9999	1166.7064	790.1796	*786.6865*	**774.6329**
M_6	Less than 0.1	9999	9999	9999	9999	9999
M_6	Less than 0.2	9999	9999	9999	9999	9999
M_6	Less than 0.3	9999	**6338.6594**	*6730.6351*	9999	9999
M_6	Less than 0.4	9999	**4146.1292**	*4169.2192*	5995.6477	9999
M_6	Less than 0.5	9999	*4146.1292*	6730.6351	4905.0506	**957.2462**
M_6	Less than 0.6	9999	*2672.9671*	3435.7925	4527.0484	**957.2462**
M_6	Less than 0.7	9999	2671.515	*1970.3431*	4527.0484	**910.5138**
M_6	Less than 0.8	9999	2671.515	**862.0898**	4527.0484	*906.7916*
M_6	Less than 0.9	9999	1930.6015	**862.0898**	4164.7177	*891.2265*
M_6	Less than 1	9999	1930.6015	**858.9613**	4164.7177	*891.2265*
M_6	Less than 1.5	9999	**831.5308**	*858.9613*	4164.7177	900.8705
M_6	Less than 2	9999	**828.2267**	*844.9996*	4164.7177	922.8004

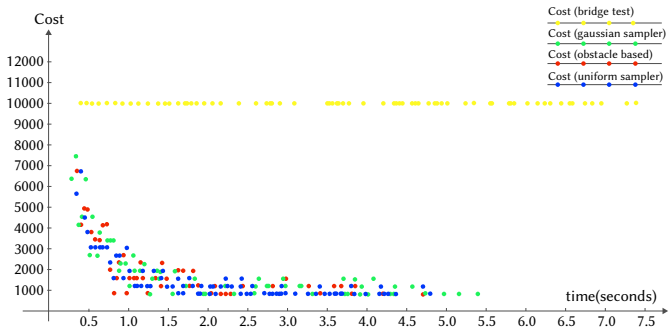

Fig. 26. Graph of cost versus time generated by the PRM variants.

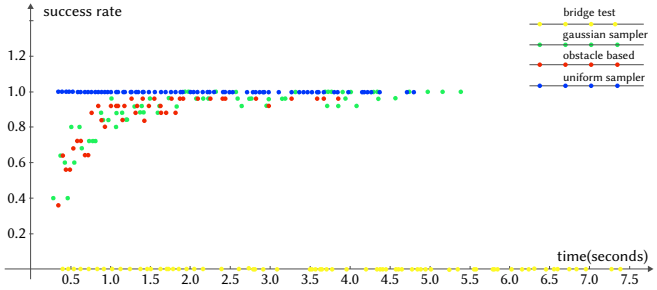

Fig. 27. Graph of success rate versus time generated by the PRM variants.

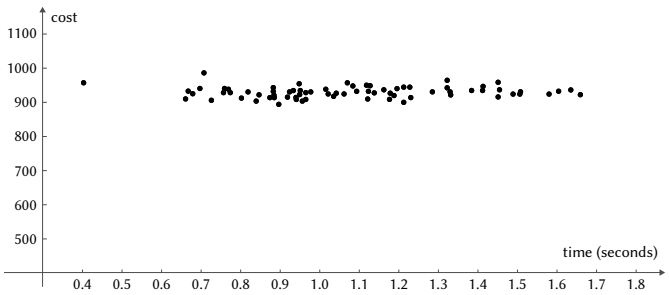

Fig. 28. Graph of cost versus time generated by the PRM variants.

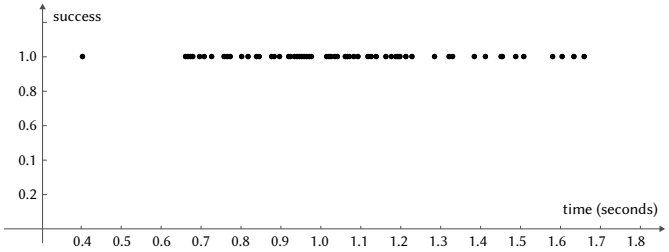

Fig. 29. Graph of success rate versus time generated by the PRM variants.

Table II and Table III shows a broader view of the experiment performed shown in Fig. 26 to Fig. 29. Both the PRM variants and

TABLE III. Success Rate Comparison of the PRM Variants and OMPRSS in Different Scenarios (Maps in Figure 19) at Various Time Steps. Source and Goal Kept at (10, 10) and (450, 450) Respectively for All Maps

Map	*Time (sec)*	*Bridge test*	*Gaussian sampler*	*Obstacle-based*	*Uniform sampler*	*OMPRSS*
M_4	*Less than 0.04*	*0*	*0*	*0*	*0*	*0*
M_4	*Less than 0.08*	*0*	*0*	*0*	*0*	*1*
M_4	*Less than 0.12*	*0*	*0*	*0*	*0*	*1*
M_4	*Less than 0.16*	*0*	*0*	*0*	*0*	*1*
M_4	*Less than 0.2*	*0*	*0*	*0*	*0*	*1*
M_4	*Less than 0.24*	*0*	*0.16*	*1*	*0*	*1*
M_4	*Less than 0.28*	*0*	*0.28*	*1*	*0*	*1*
M_4	*Less than 0.32*	*0*	*0.28*	*1*	*0*	*1*
M_4	*Less than 0.36*	*0*	*0.48*	*1*	*0*	*1*
M_4	*Less than 0.4*	*0*	*0.48*	*1*	*1*	*1*
M_5	*Less than 0.01*	*0*	*0*	*0*	*0*	*0*
M_5	*Less than 0.06*	*0*	*0*	*0*	*0*	*1*
M_5	*Less than 0.11*	*0*	*0*	*0*	*0*	*1*
M_5	*Less than 0.16*	*0*	*0*	*0*	*0*	*1*
M_5	*Less than 0.21*	*0*	*0*	*0*	*0*	*1*
M_5	*Less than 0.26*	*0*	*0*	*0*	*0*	*1*
M_5	*Less than 0.31*	*0*	*0.68*	*0.96*	*0*	*1*
M_5	*Less than 0.36*	*0*	*0.96*	*1*	*0*	*1*
M_5	*Less than 0.41*	*0*	*0.96*	*1*	*1*	*1*
M_5	*Less than 0.5*	*0*	*0.96*	*1*	*1*	*1*
M_6	*Less than 0.1*	*0*	*0*	*0*	*0*	*0*
M_6	*Less than 0.2*	*0*	*0*	*0*	*0*	*0*
M_6	*Less than 0.3*	*0*	*0.4*	*0.36*	*0*	*0*
M_6	*Less than 0.4*	*0*	*0.64*	*0.64*	*0.44*	*0*
M_6	*Less than 0.5*	*0*	*0.64*	*0.68*	*0.56*	*1*
M_6	*Less than 0.6*	*0*	*0.8*	*0.72*	*0.6*	*1*
M_6	*Less than 0.7*	*0*	*0.8*	*0.88*	*0.6*	*1*
M_6	*Less than 0.8*	*0*	*0.8*	*1*	*0.64*	*1*
M_6	*Less than 0.9*	*0*	*0.88*	*1*	*0.64*	*1*
M_6	*Less than 1*	*0*	*0.88*	*1*	*0.64*	*1*
M_6	*Less than 1.5*	*0*	*1*	*1*	*0.64*	*1*
M_6	*Less than 2*	*0*	*1*	*1*	*0.64*	*1*

OMPRSS were executed in three maps M_4, M_5 and M_6 given in Fig. 19, the location of the source and the goal kept at (10, 10) and (450, 450) respectively. The planners were executed 25 times per iteration of increasing sample points and the average of both cost and the success rate were recorded and arranged in certain time steps as shown in Table II and Table III for all the planners.

V. Discussion

The experimental results of OMPRSS show some intriguing outcomes. The initial experiment shows the basic working of OMPRSS in which we can observe that though the path length is not optimal, the time of planning is adequate. It also somehow tends not to waste time on C_{free} areas rather focus on obstacle dominated areas. OMPRSS has the flexibility of changing various parameters which includes the range factor which determines the area of generating the random sample points and the space for increasing or decreasing the number of sample points and the depth limit of recursion. These parameters open up more dimensions for experiments. From the theoretical point of view, though the sample points are generated randomly, the larger the area of generating points (larger range factor) the chance of getting a larger path length is higher and this holds for the other way around as well. The experiment of changing the range factor was performed as to view whether this condition would hold. Table I shows the result of changing the range factor in which cost generated by R_3 is almost always higher compared to R_1 and R_2. But there is a better part of R_3. We see this in *Sl. No.* 16 to 18 of Table I. In M_3, for source at (100, 70) and goal at (230, 200), R_1 and R_2 fails to find the path, whereas R_3 does. This shows that R3 can work better in cases where the destination is augmented by obstacles, for example, dealing with concave obstacles. On the other hand, altering the number of sample points and the depth of recursion can result in different responses. Theoretically, increasing the sample points in OMPRSS can increase the chances of getting a successful planning faster but we can see in Fig. 16 that time really does not decrease with increasing sample points, rather fluctuates. The cost also does not either increase or decrease with increasing sample points (Fig. 15). In Fig. 16, the worst case of time required for planning is about 0.175 seconds and below 0.01 seconds at best. As OMPRSS takes fairly short time for planning, the planner can be executed multiple times and the best cost can be considered. On the contrary, increasing the depth is worse than increasing the sample points. We see this in Fig. 17 and Fig. 18 where the cost on average is higher whereas the time required for planning has the probability of shooting up high with increasing depth.

The final stage of the experiment was the comparison of OMPRSS with the variants of PRM. In Fig. 20, the PRM variants start with a higher cost which gradually decreases with increasing sample points S_i, except for Bridge test PRM. The Gaussian sampler and Obstacle based gradually gets higher success rate with S_i and the uniform sampler has a consistent success rate (Fig. 22). The time on the other hand increases with S_i (Fig. 21). Whereas for OMPRSS, neither the cost and time increases or decreases with S_i, rather oscillate with a consistent success rate. And from Fig. 26 to Fig. 29 and Table II and Table III we can observe that OMPRSS tends to converge faster with adequate cost. In all the experiments, the Bridge test PRM failed in most cases as it was developed to generate sample points only at narrow corridors, due to which a single connected roadmap could not be constructed.

VI. Conclusion

This paper introduces a new approach of a deliberative sampling-based motion planning which tries to find a path from source to destination expectantly using randomly generated collision free sample points and straight lines. When the straight-line path from source to goal is collision prone, the planner tries to find the path around the obstacles recursively resulting in a planning which focuses more on obstacle outweighing areas and spending lesser time on the collision free areas. Comparison with PRM variants shows that the proposed planner gives solutions at a faster rate.

Though OMPRSS can deliver fast planning, there are downsides to the algorithm. OMPRSS mostly fails in highly obstacle-dominant maps and to plan in such maps, higher depth of recursion is required. But the planner can produce lots of sample points which can lead to unnecessary turns and also take a large amount of time when it tries to find a path around an obstacle especially when the depth is high. In addition, OMPRSS is not designed for dynamic environments. The algorithm also tends to struggle in narrow corridors especially if the length of the corridor is large. In the experiments performed, OPMRPSS is not tested in high dimensional spaces and also not tested on real robots.

Optimistic motion planning using recursive sub-sampling is at its initial phase, as a result it still has some drawbacks, like the path length can be high and somewhat far from being optimal, finding path in obstacle dominant areas can take a large amount of time and having to choose ideal parameters for different scenarios is still done manually. In addition, the experiments were performed in 2D environments using a point robot. Implementing the OMPRSS in higher dimensional environments and also addressing to the drawbacks mentioned above are the future perspectives for improving the proposed approach.

Acknowledgement

This work is funded by the Indian Institute of Information Technology, Allahabad.

References

[1] Latombe, J.-C., "Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts," *The International Journal of Robotics Research*, 1999, 18, (11), pp. 1119-1128. doi:10.1177/02783649922067753.

[2] Kavraki, L. and Latombe, J., "Randomized Preprocessing of Configuration for Fast Path Planning," in, *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, (1994)), pp. 2138-2145 vol.2133. doi:10.1109/ROBOT.1994.350966.

[3] Elbanhawi, M. and Simic, M., "Sampling-Based Robot Motion Planning: A Review," *IEEE Access*, 2014, 2, pp. 56-77. doi:10.1109/ACCESS.2014.2302442.

[4] Lavalle, S.M., "Rapidly-Exploring Random Trees: A New Tool for Path Planning," October 1998, Technical Report,Computer Science Dept., I.S.U. (TR 98−11). doi: 10.1.1.35.1853.

[5] Tsianos, K.I., Sucan, I.A., and Kavraki, L.E., "Sampling-Based Robot Motion Planning: Towards Realistic Applications," *Computer Science Review*, 2007, 1, (1), pp. 2-11. doi:10.1016/j.cosrev.2007.08.002.

[6] Kumar, A. and Kala, R., "Linear Temporal Logic-Based Mission Planning," *International Journal of Interactive Multimedia and Artificial Intelligence*, 2016, 3. doi:10.9781/ijimai.2016.375.

[7] Montana, F.J., Liu, J., and Dodd, T.J., "Sampling-Based Reactive Motion Planning with Temporal Logic Constraints and Imperfect State Information," in Petrucci, L., Seceleanu, C., and Cavalcanti, A. (eds.), *Critical Systems: Formal Methods and Automated Verification*, (Springer International Publishing, 2017)), pp. 134-149. doi:10.1007/978-3-319-67113-0_9.

[8] Kavraki, L.E., Svestka, P., Latombe, J., and Overmars, M.H., "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, 1996, 12, (4), pp. 566-580. doi:10.1109/70.508439.

[9] Amato, N.M. and Wu, Y., "A Randomized Roadmap Method for Path and Manipulation Planning," in, *Proceedings of IEEE International Conference on Robotics and Automation*, (1996)), 1, pp. 113-120 vol.111. doi:10.1109/ROBOT.1996.503582.

[10] Persson, S.M. and Sharf, I., "Sampling-Based a* Algorithm for Robot Path-Planning," *The International Journal of Robotics Research*, 2014, 33, (13), pp. 1683-1708. doi:10.1177/0278364914547786.

[11] Sucan, I.A., Moll, M., and Kavraki, L.E., "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, 2012, 19, (4), pp. 72-82. doi:10.1109/MRA.2012.2205651.

[12] Bohlin, R. and Kavraki, L.E., "Path Planning Using Lazy Prm," in, *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, (2000)), 1, pp. 521-528 vol.521. doi:10.1109/ROBOT.2000.844107.

[13] Karaman, S. and Frazzoli, E., "Sampling-Based Algorithms for Optimal Motion Planning," *The International Journal of Robotics Research*, 2011, 30, (7), pp. 846-894. doi:10.1177/0278364911406761.

[14] Dobson, A. and Bekris, K.E., "Sparse Roadmap Spanners for Asymptotically near-Optimal Motion Planning," *The International Journal of Robotics Research*, 2014, 33, (1), pp. 18-47. doi:10.1177/0278364913498292.

[15] Hsu, D., Tingting, J., Reif, J., and Zheng, S., "The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners," in, *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, (2003)), 3, pp. 4420-4426 vol.4423. doi:10.1109/ROBOT.2003.1242285.

[16] Boor, V., Overmars, M.H., and Stappen, A.F.v.d., "The Gaussian Sampling Strategy for Probabilistic Roadmap Planners," in, *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, (1999)), 2, pp. 1018-1023 vol.1012. doi:10.1109/ROBOT.1999.772447.

[17] Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C., and Vallejo, D., "Obprm: An Obstacle-Based Prm for 3d Workspaces," in, *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics: the algorithmic perspective: the algorithmic perspective*, (A. K. Peters, Ltd., 1998 of Conference, Edition edn.), pp. 155–168. doi:10.1201/9781439863886-19.

[18] Geraerts, R. and Overmars, M.H., "Reachability-Based Analysis for Probabilistic Roadmap Planners," *Robotics and Autonomous Systems*, 2007, 55, (11), pp. 824-836. doi:10.1016/j.robot.2007.06.002.

[19] Geraerts, R. and Overmars, M.H., "Sampling and Node Adding in Probabilistic Roadmap Planners," *Robotics and Autonomous Systems*, 2006, 54, (2), pp. 165-173. doi:10.1016/j.robot.2005.09.026.

[20] Kuffner, J.J. and LaValle, S.M., "Rrt-Connect: An Efficient Approach to Single-Query Path Planning," in, *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, (2000)), 2, pp. 995-1001 vol.1002. doi:10.1109/ROBOT.2000.844730.

[21] Kala, R., "Rapidly Exploring Random Graphs: Motion Planning of Multiple Mobile Robots," *Advanced Robotics*, 2013, 27, (14), pp. 1113-1122. doi:10.1080/01691864.2013.805472.

[22] Janson, L., Ichter, B., and Pavone, M., "Deterministic Sampling-Based Motion Planning: Optimality, Complexity, and Performance," *The International Journal of Robotics Research*, 2017, 37, (1), pp. 46-61. doi:10.1177/0278364917714338.

[23] Solovey, K. and Kleinbort, M., "The Critical Radius in Sampling-Based Motion Planning," *The International Journal of Robotics Research*, 2019, 39, (2-3), pp. 266-285. doi:10.1177/0278364919859627.

[24] Ichter, B., Schmerling, E., Lee, T.W.E., and Faust, A., "Learned Critical Probabilistic Roadmaps for Robotic Motion Planning," in, *2020 IEEE International Conference on Robotics and Automation (ICRA)*, (2020)), pp. 9535-9541. doi:10.1109/ICRA40945.2020.9197106.

[25] Vonásek, V., Pěnička, R., and Kozlíková, B., "Computing Multiple Guiding Paths for Sampling-Based Motion Planning," in, *2019 19th International Conference on Advanced Robotics (ICAR)*, (2019)), pp. 374-381. doi:10.1109/ICAR46387.2019.8981589.

[26] Kim, D., Kwon, Y., and Yoon, S., "Adaptive Lazy Collision Checking for Optimal Sampling-Based Motion Planning," in, *2018 15th International Conference on Ubiquitous Robots (UR)*, (2018)), pp. 320-327. doi:10.1109/URAI.2018.8442203.

[27] Švestka, P. and Overmars, M.H., "Coordinated Path Planning for Multiple Robots," *Robotics and Autonomous Systems*, 1998, 23, (3), pp. 125-152. doi:10.1016/S0921-8890(97)00033-X.

[28] Clark, C.M., "Probabilistic Road Map Sampling Strategies for Multi-Robot Motion Planning," *Robotics and Autonomous Systems*, 2005, 53, (3), pp. 244-264. doi:10.1016/j.robot.2005.09.002.

[29] Yao, Z. and Gupta, K., "Path Planning with General End-Effector Constraints," *Robotics and Autonomous Systems*, 2007, 55, (4), pp. 316-327. doi:10.1016/j.robot.2006.11.004.

[30] Kala, R., "Sampling Based Mission Planning for Multiple Robots," in, *IEEE Congress on Evolutionary Computation (CEC)*, (2016)), pp. 662-669. doi:10.1109/CEC.2016.7743856.

Lhilo Kenye

Lhilo Kenye received his B.Tech. degree in Information Technology from School of Engineering and Technology, Nagaland University, India in 2014. He received his M.Tech. degree in Information Technology, specialization in Robotics from Indian Institute of Information Technology, Allahabad, India in 2016. He is currently pursuing PhD in Information technology, specialization in Robotics, from Indian Institute of Information Technology, Allahabad, India. His current work is mainly in the field of robot navigation and computer vision.

Rahul Kala

Rahul Kala received the B.Tech. and M.Tech. degrees in Information Technology from the Indian Institute of Information Technology and Management, Gwalior, India in 2010. He received his Ph.D. degree in cybernetics from the University of Reading, UK in 2013. He is currently working as an Assistant Professor in the Indian Institute of Information Technology, Allahabad, India. He is the author of four books and over 100 papers. He is a recipient of the Early Career Research Grant from the Department of Science and Technology; Best PhD dissertation award from the IEEE ITS Society; and Commonwealth scholarship from the British Government.