2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Autonomous Robotic Exploration Based on Multiple Rapidly-exploring Randomized Trees

Hassan Umari[1] and Shayok Mukhopadhyay[2]

Abstract— **Efficient robotic navigation requires a predefined map. Various autonomous exploration strategies exist, which direct robots to unexplored space by detecting frontiers. Frontiers are boundaries separating known space form unknown space. Usually frontier detection utilizes image processing tools like edge detection, thus limiting it to two dimensional (2D) exploration. This paper presents a new exploration strategy based on the use of multiple Rapidly-exploring Random Trees (RRTs). The RRT algorithm is chosen because, it is biased towards unexplored regions. Also, using RRT provides a general approach which can be extended to higher dimensional spaces. The proposed strategy is implemented and tested using the Robot Operating System (ROS) framework. Additionally this work uses local and global trees for detecting frontier points, which enables efficient robotic exploration. Current efforts are limited to the single robot case. Extension to multi-agent systems and three-dimensional (3D) space is left for future effort.**

I. INTRODUCTION

The main goal of an autonomous robotic exploration algorithm is to direct robots to unknown space, thus expanding the known and explored portion of a map which is being created as a robot moves. Frontier based exploration strategies [1], [2], [3] which are usually used for robotic exploration, direct robots to frontier edges. Frontier edges are lines that separate known space form unknown space in an occupancy grid map. Once a frontier edge is detected, a point on the detected edge, which is normally the centroid, is assigned to a robot for exploration. In order to extract frontier edges, the entire map has to be processed, and as the map expands, processing it will consume more and more computational resources [4]. This has led to research on efficient detection of frontier edges [4], [5].

The second branch of exploration strategies deploy randomized search techniques such as the simple random walker, and the Sensor-based Random Tree (SRT) [6], [7], which is a variation of RRT [8]. RRT is a path planning algorithm that samples space using randomly generated points. Random points are used to extend edges in a tree-like structure, which consists of nodes and edges. One possible mode of RRT based exploration is to make robots follow the above mentioned tree structure as the tree structure grows [9].

RRT is heavily biased towards unexplored and unvisited regions. In addition, RRT provides completeness [8], which ensures complete map coverage. Due to these properties, RRT has gained interest in exploration. However, making robots use RRT can be inefficient because of the possibility that robots may try to revisit map areas that are already explored (i.e. '*overlapping*' may occur). This is because branches can grow from random points at different time-steps. To avoid this, SRT was used in [5]. SRT grows one branch at time, a robot follows this branch in space until the branch can no longer extend due to the existence of a physical obstacle to robot motion. However, this approach of SRT doesn't necessarily reduce overlapping either. If a branch is not able to extend, the robot has to go back and track previous positions in an attempt to find a position in space, where a new branch can extend from. This process is known as backtracking, and is a major source for overlapping. As a result, researchers proposed solutions to reduce backtracking [10], [11].

This paper presents a new strategy for detecting frontier points using RRT. The robot is not made to follow a growing RRT-tree physically in space. Instead, the tree is used in the search for frontier points, and this search runs independently from robot movement. The detected points are filtered and queued to be assigned to a robot. When a point is assigned to a robot for exploration, the robot moves towards the assigned point. During this process, sensors onboard the robot (e.g. laser scanner) scan and explore neighboring areas within the sensor range. An additional novelty of this work is the use of multiple independently growing trees for speeding up the search for frontier points.

The exploration strategy is presently implemented on a single robot using ROS framework. However, it can be extended to multiple robots and to 3D space. The exploration strategy is tested in presence of other commonly used navigation components, in particular, the simultaneous localization and mapping (SLAM) module [12], [13] and the path planner module. The good performance of exploration seen in this work provides encouragement for using RRT based exploration approaches for fast path planning and exploration in higher dimensional spaces, because this can help extend the work in [14], [15] for computing invariant sets of N-dimensional systems via efficient RRT based approaches.

This paper is organized as follows. Section II provides necessary terminology used in the description of the proposed exploration strategy. Section III introduces the exploration strategy and its components. Section IV discusses strategy implementation. Section V, shows the simulation and experimental setup. Finally, simulation and experimental results are shown in Section VI, and the paper is concluded in Section

[1]Hassan Umari is a graduate (M.Sc.) student in the Mechatronics Engineering program, American University of Sharjah, 26666 Sharjah, UAE b00062945@aus.edu

[2]Shayok Mukhopadhyay is an assistant professor of Electrical Engineering, American University of Sharjah, 26666 Sharjah, UAE smukhopadhyay@aus.edu

VII.

II. Preliminary Terminology

Most of the following definitions are related to the RRT algorithm [16].

Map X: The set of total space i.e. the set which contains occupied (obstacles), unoccupied (free) space, and the unknown (unexplored) space.

Occupancy grid: A 2D map representation that divides the map into cells, each cell is indexed by its coordinates, and a cell has a value of 1, 0, or -1 representing whether it is occupied (i.e. there is an obstacle), free, or unknown respectively.

Free Space X_{free}: A set representing free space, i.e. the space that has been explored and is not occupied by obstacles.

Unknown region: An unexplored (unknown) subset of the map.

Vertices V: Nodes or points in a map, in RRT these points are connected to one another by the tree branches (edges), each point on the tree is a vertex, vertices are stored in the set V.

Edge E: An edge is the branch or line connecting two vertices, each edge is stored in terms of the spatial coordinates of the two connected points, edges are stored in the edge set E.

Graph G: Edges and vertices both form a graph, $G = (V, E)$, in RRT the graph has a tree structure.

SampleFree: A function that returns random points that are independently identically distributed (i.i.d) in the map X.

Nearest ($G = (V, E), x \subset X_{free}$): A function that takes a graph (i.e. tree vertices and edges) and a point x in the free space as inputs, and returns a point $v \subset V$ such that, $\texttt{Nearest}(G = (V, E), x) = argmin_{v \subset V} \|x - v\|$. If there are multiple points $v \in V$, which are at a minimum distance from x, then the first point v encountered while computing $argmin_{v \subset V} \|x - v\|$ is returned.

Steer: Is a function that takes two points x, y, and returns a point z, where $\|z - y\|$ is minimized, while $\|z - x\| \leq \eta$, for an $\eta > 0$, η is the tree growth rate. Large value of η corresponds to a faster tree growth (i.e tree expands faster).

GridCheck: A function that takes a map and two points. It returns 0 if there is an obstacle between the points based on the given map. It returns -1 if there is an unknown region between the points. Otherwise it returns 1 indicating that the points are in the known free space.

PublishPoint: A function that sends detected frontier points to the filter. The 'filter' is described later in the paper.

Old frontier point: A frontier point detected in earlier iteration of the proposed frontier detector, and which is no longer in the unknown region of the map.

Invalid frontier point: A frontier point the robot cannot physically reach i.e. no valid path exists between robot's position and the given frontier point.

III. Strategy Description

The exploration strategy is split into three modules; the RRT-based frontier detector module, the filter module, and the robot task allocator module. The frontier detector is responsible for detecting frontier points and passing them to the filter module. The filter module clusters the frontier points and stores them, in this work the mean shift [17] clustering algorithm is used. The filter module also deletes invalid and old frontier points. The task allocator module receives the clustered frontier points from the filter module, and assigns them to a robot for exploration. Each module is explained in more details in Subsections III-A, III-C, and III-D. The exploration strategy also requires mapping and path planning modules, which are available in existing work [12], [13], [18]. An overall high level schematic diagram of the exploration strategy is shown in Fig. 1. Implementation details are available in Section IV.

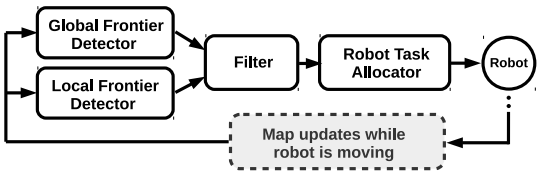


Fig. 1: Overall schematic diagram of the exploration algorithm

The proposed configuration, i.e. splitting the exploration strategy into three modules, i.e. 'frontier detector module, filter module, and task allocator module' has the following advantages. The task allocation routine can be changed without affecting the detection of frontier points. Similarly different types of frontier detectors could be tested without affecting the behavior of the task allocator. Additionally, multiple instances of the frontier detector can be run in parallel for faster frontier detection, if needed.

A. RRT-Based Frontier Detector Module

The RRT-based frontier detector module discovers frontier points. In our work, a point that is reached by the growing RRT tree is considered a frontier point, if this point lies in the unknown region of the map. In our implementation, the map is represented as an occupancy grid, points located in the unknown region carry a cell value of -1, so by reading the cell value of a point, it can be classified as unknown, free or occupied (i.e. an obstacle exists). Please note that in this work the initial occupancy grid map is filled only with unknown cells (obstacles are not known). Obstacles and known regions (cells) are marked (i.e. the cell values in the occupancy grid are updated) as a robot explores the map.

We propose the use of two versions of frontier detectors: i.) a local frontier detector, and ii.) a global frontier detector. The details related to these are provided below.

1) Local Frontier Detector:

An outline of the local frontier detector is listed in Algorithm 1. Similar to the RRT algorithm, it starts from a single initial vertex $V = \{x_{init}\}$, and the edge set $E = \phi$, at each iteration a random point $x_{rand} \subset X_{free}$ is sampled. The first vertex of the tree which is nearest to x_{rand} is found, this point is called $x_{nearest} \subset V$. Then, the **Steer** function

Algorithm 1 Local Frontier Detector

1: $V \leftarrow x_{init}$; $E \leftarrow \phi$;
2: **while** `True` **do**
3: $x_{rand} \leftarrow$ `SampleFree`;
4: $x_{nearest} \leftarrow$ `Nearest`$(G(V,E), x_{rand})$;
5: $x_{new} \leftarrow$ `Steer`$(x_{nearest}, x_{rand}, \eta)$;
6: **if** `GridCheck`$(map, x_{nearest}, x_{new}) = -1$ **then**
7: `PublishPoint`(x_{new});
8: $V \leftarrow x_{current}$; $E \leftarrow \phi$; ▷ reset the tree
9: **else if** `GridCheck`$(map, x_{nearest}, x_{new}) = 1$ **then**
10: $V \leftarrow V \cup \{x_{new}\}$; $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$;
11: **end if**
12: **end while**

generates a point x_{new}. The **GridCheck** function checks if x_{new} lies in the unknown region, or if any point of the line segment between x_{new} and $x_{nearest}$ lies in the unknown region. If either of the above conditions is true, then x_{new} is considered as a frontier point. The point x_{new} is then sent to the filter module, and the tree is reset, i.e. tree vertices and edges are deleted. The next iteration of the tree starts from the current robot position (i.e. $V = \{x_{current}\}$, and $E = \phi$). This step is shown in line 8 in Algorithm 1. If there is no obstacle at x_{new} and no obstacle in the space between x_{new} and $x_{nearest}$, the tree extends by adding x_{new} as a new vertex. An edge is created between x_{new} and $x_{nearest}$.

The resetting of the tree as shown in line 8 in Algorithm 1 is one of the major differences between the usage of RRT for exploration in this work, compared to other standard implementations of RRT available in literature. The reason behind resetting the tree in the local detector is explained in Subsection III-A.3.

For each robot running the local frontier detector, a tree generates by the process described above. Once the tree reaches an unknown region, a frontier point is marked and the tree is reset. This process happens during a robot's motion, therefore the tree grows from a new initial point each time it resets according to line 8 in Algorithm 1. The local detector is proposed for fast detection of frontier points in the immediate vicinity of the robot at any time. Figure 2 shows the propagation of local RRT and the process of detecting a frontier point.

2) Global Frontier Detector:

The outline for the global frontier detector is identical to the local frontier detector's outline (Algorithm 1), except that line 8 is removed. The tree doesn't reset and keeps growing during the whole exploration period (i.e. until the map is completely explored), which makes the global frontier detector algorithm similar to RRT. The global frontier detector is meant to detect frontier points through the whole map and in regions far from the robot.

In future work related to multi-robot based exploration, every robot can run its own local frontier detector, and a master robot can additionally run a global frontier detector. The scope of this paper is limited to single robot exploration, hence, both frontier detectors are run by the same robot.

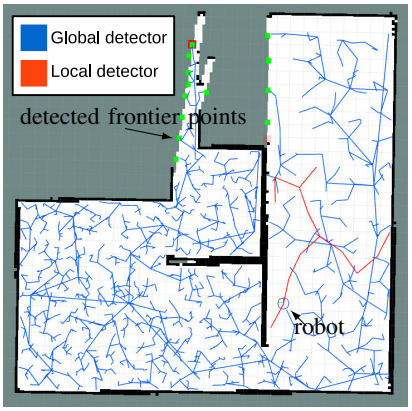


Fig. 3: Global and local frontier detectors

3) The Need for Global and Local Frontier Detectors:

As explained above, the local tree is reset after it detects a frontier point and starts growing again from the robot's current position. This has two consequences: 1. It allows detection of frontier points quicker because, the robot is always given a path to reach a frontier point, so if the tree starts growing from the robot's current position, the chance that the next point picked from the RRT for exploration lies in the unknown space is higher. 2. The robot can miss exploring small corners in a map. To fix the problem of missing exploration of corners, and also to make sure that points which are far from the robot's current position are detected and explored, we use the global frontier detector.

However, the growth of the tree in the global frontier detector becomes slower as the tree grows larger (i.e. the number of tree vertices increases). This can be explained by analyzing the Voronoi diagram of RRT, as the number of tree vertices increases, the space is decomposed into smaller and smaller Voronoi regions. As a result, the **steer** function will create edges of smaller length, hence, detection of frontier points also becomes slower. This is why the local frontier detector is needed. Thus our proposed strategy uses local and global RRT-based frontier detectors to complement one another so that frontier detection is as fast as possible.

B. Why RRT?

RRT is heavily biased to grow towards unknown regions of the map [8], which biases the tree to detect frontier points. This property is explained using Fig. 4, which shows the Voronoi diagram of an RRT tree during exploration. Vertices with larger Voronoi regions are closer to the unknown space (as the selection of vertices is based on finding the nearest neighbor [8], the tree is more likely to extend from these regions).

RRT is also not limited to 2D space, as a result it can be used in 3D exploration (which can help to extend the work in [14], [15]). One possible 3D map representation is the OctoMap [19]. OctoMap provides a 3D occupancy grid representation of the environment, where it maps occupied space, free space, and unknown space. Whether the map is represented as a 2D occupancy grid or a 3D occupancy grid,

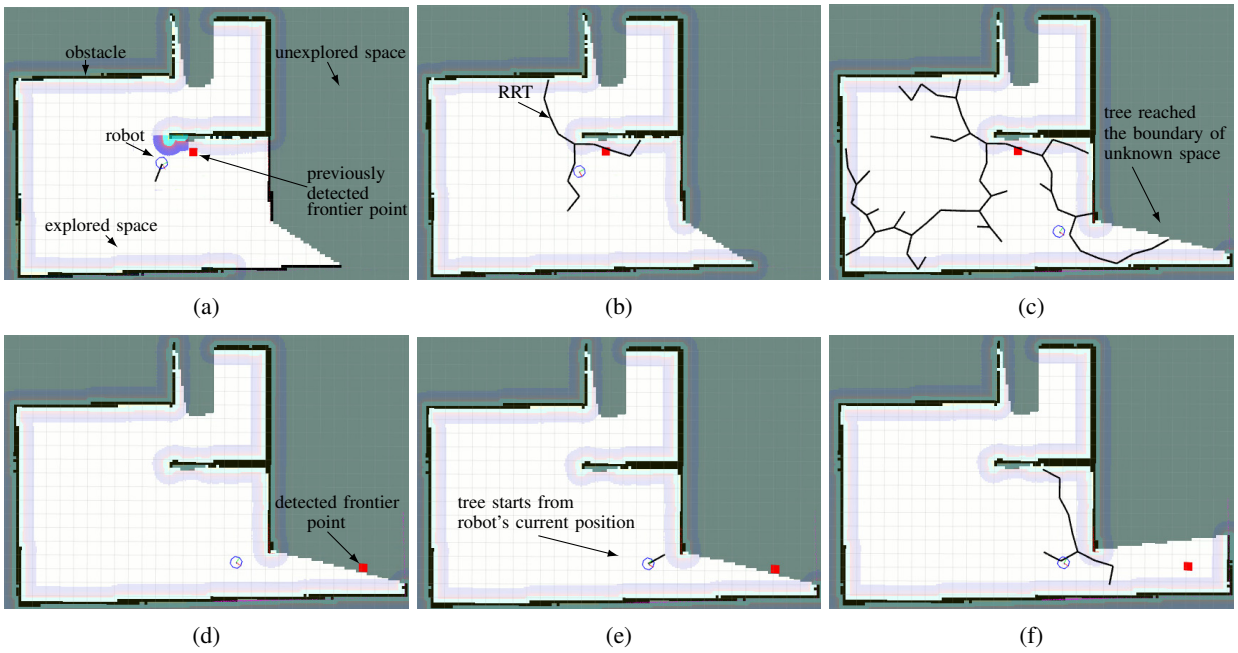


Fig. 2: Propagation of the local RRT and the detection of frontier points

In (a) the tree starts from the current position of the robot, in (b) and (c) the tree keeps growing, in (d) a tree vertex lying in the unknown region is marked as a frontier point and the tree is reset. In (e) and (f) the loop repeats where the tree grows back again from the robot's current position

The **GridCheck** function only checks for the x_{new} cell, and the cells that are on the line segment between x_{new} and $x_{nearest}$ in the occupancy grid. Here x_{new} corresponds to the vertex found using the **Steer** function, and $x_{nearest}$ is a vertex in the RRT graph. Additionally, RRT is also probabilistically complete [20], so it is guaranteed that the map will be completely discovered and explored.

C. The Filter Module

The filter module receives the detected frontier points from all the local frontier detectors, and from the global frontier detector. The filter module first clusters the points, and it stores only the center of each cluster, the remaining points are discarded (not stored). The clustering and subsequent discarding process is needed to reduce the number of frontier points, because global and local frontier detectors can provide too many frontier points which are extremely close to each other. If such points are sent to the robot task allocator module, then there will be unnecessary consumption of computational resources, and no additional information about the map is necessarily gained. The filter module also deletes invalid and old frontier points in each iteration.

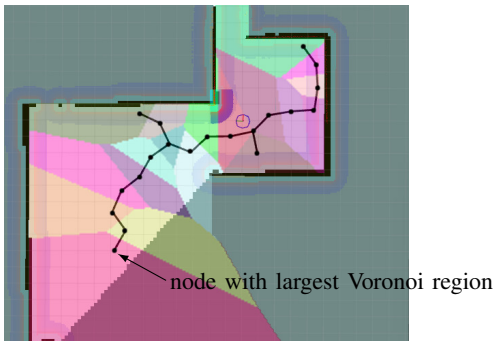


Fig. 4: Voronoi diagram of RRT

D. Robot Task Allocator Module

This module receives filtered frontier points from the filter module and assigns them to a robot. The design of the robot task allocator module is similar to [21]. The robot task allocator module assigns frontier points to be explored by a particular robot by considering the following:

Navigation cost (N): It is defined as the expected distance to be traveled by a robot to reach a frontier point. In order to simplify computation, the navigation cost is calculated by considering the norm of the difference between a robot's current position and the location of a frontier point.

Information gain (I): It is defined as the area of unknown region expected to be explored for a given frontier point. The information gain is quantified by counting the number of unknown cells surrounding a frontier point within a user defined radius. This radius is referred to as the information gain radius, which should be set to a value equal to the perception sensor range. The area is then calculated by multiplying the number of cells within the information gain radius, by the area of each cell (which is computed from the map resolution). In Fig. 5, the information gain approximately equals to $1.81\ m^2$ (i.e. number of unknown cells is 181, each cell is a square with a width equal to map resolution,

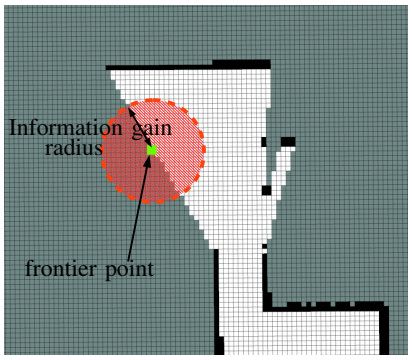


Fig. 5: Information gain region of a frontier point

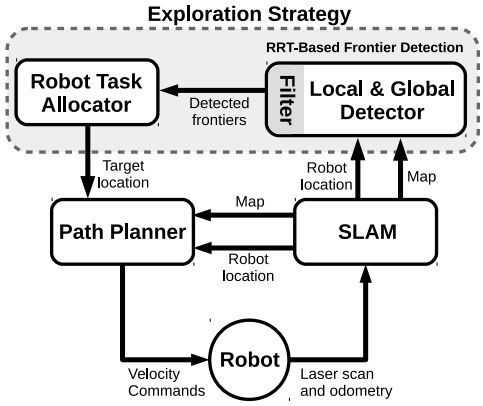


Fig. 6: Implementation diagram

which in this case, is 0.1 m).

Revenue from a frontier point (R): For a given frontier point x_{fp}, and a current robot location x_r, the revenue R obtained from exploring x_{fp}, is calculated as:

$$R(x_{fp}) = \lambda h(x_{fp}, x_r) I(x_{fp}) - N(x_{fp}), \tag{1}$$

$$h(x_{fp}, x_r) = \begin{cases} 1, & \text{if } ||x_r - x_{fp}|| > h_{rad} \\ h_{gain}, & h_{gain} > 1 \end{cases} \tag{2}$$

where λ is a positive user-defined constant which is used as a weight. The weight λ is used to give more importance to the information gained from exploring a frontier point, compared to the navigation cost. This also helps make the magnitudes of the terms I, and N have a similar order of magnitude. Also $h(x_{fp}, x_r)$ is the hysteresis gain [21] and is calculated by using Equation 2, it equals unity if the given frontier point x_{fp} is outside a certain radius h_{rad} from the robot's current location x_r. The positive number h_{rad} is set based on user experience. And $h(x_{fp}, x_r) = h_{gain},\ h_{gain} > 1$ if the given frontier point is within a certain radius h_{rad} form the robot's current location x_r. Please note that h_{gain} should be set larger than 1, so that the robot is biased to explore frontier points in its vicinity; the advantage of this is that it avoids overlapping.

For each frontier point, a revenue R is calculated by using Equation 1. The point with the highest revenue is assigned to the robot for exploration.

IV. Implementation

Figure 6 shows how the exploration strategy described above is implemented. The exploration strategy consists of the SLAM module, path planning module, global and local frontier detector modules, the filter module, and the robot task allocator module. Different ready-made ROS packages are used in the implementation for mapping and path planning. Also the proposed exploration strategy is itself implemented as a ROS package consisting of four nodes; the local frontier detector node, the global frontier detector node, the filter node, and the robot task allocator node.

The ROS 'gmapping' package is used for generating the map and localizing the robot. The 'gmapping' package implements a SLAM algorithm that uses a Rao-Blackwellized particle filter [12], [13].

The ROS Navigation stack is used to control and direct the robot towards exploration goals (i.e. the assigned frontier points x_{fp}). Path planning based on the A* algorithm [18] is one of the packages available inside the ROS navigation stack, which is used in our work for planning paths to an assigned frontier point from the robot's current position.

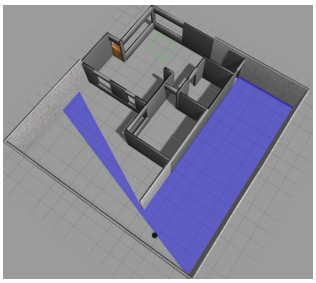
The global and local frontier detectors are programmed as ROS nodes written in C++. Every local and global frontier detector publishes detected frontier points on a common ROS topic. The filter node subscribes to this topic, so that it can receive all detected frontier points. The filter node processes the received frontier points, as described in Subsection III-C, and then publishes remaining valid frontier points on a ROS topic which is subscribed by the robot task allocator node. The robot task allocator node receives points provided by the filter node and assigns them for exploration as described in Subsection III-D. For details related to ROS terminology (publish, subscribe, etc.) please see [22], and for specific details related to our implementation please see [23].

V. Simulation and Experimental Setup

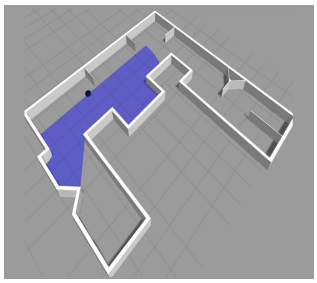
The proposed RRT-based local and global frontier detectors are compared against an image processing-based frontier detector. The same robot task allocator (explained above) is used in both cases. Also, the **steer** function used in RRT-based exploration (as shown in line 5 of Algorithm 1) requires tree growth rate η as an argument. Below we present two simulation maps and one experimental map which are explored using our proposed exploration strategy. For each map, we perform a total of 70 exploration runs. Out of these 70, 10 exploration runs are performed using an image processing-based frontier detector, the remaining 60 runs are performed using our proposed local and global frontier detectors. Further, these 60 exploration runs are divided into 6 sets of 10 exploration runs, where the global frontier detector in each set uses a **steer** function with a particular growth rate η, where $\eta \in \{0.5, 1, 4, 6, 10, 15\}$, and the local frontier detector uses a **steer** function with fixed $\eta = 1$.

A. Simulation Setup

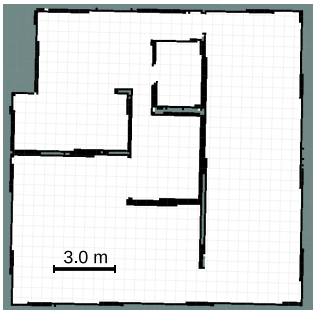
Simulations were carried out using the Gazebo simulator [24], which provides realistic robotic movements, a physics



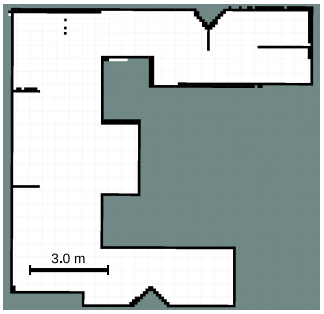
(a) First map (large simulation environment)



(b) Second map (small simulation environment)



(c) Occupancy grid generated for the first simulation environment



(d) Occupancy grid generated for the second simulation environment

Fig. 7: Simulation environments

engine, and the generation of sensor data combined with noise.

1) Environments Used In Simulations:

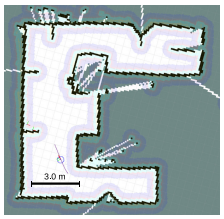
Two environments are used for simulation. The first environment, shown in Fig. 7a, is a large map with an area of approximately $182\ m^2$ (free space area), and the robot's radius is $0.175\ m$. In the experiments made using this environment, the laser scanner range is set to $50\ m$.

The second environment, shown in Fig. 7b, is a small map made very similar to the real map that is actually used in the real experimental setup, the area of the map is approximately $49\ m^2$. In the experiments made using this environment, the laser scanner range is set to $5\ m$ which is similar to the range of the actual laser scanner (i.e. $4\ m$) used in the real setup. We use two sizes of maps and two different laser scanner ranges, in order to observe the effect of map size and laser scanner range on the proposed exploration strategy. The outcome of each exploration experiment is an occupancy gird. Figure 7c and Fig. 7d show the occupancy grids obtained for the simulated environments.

B. Experimental Setup

The mobile robot platform used in the experiments is the Kobuki base. For perception, the Hokuyo URG-04LX laser range scanner is used [25], it provides range measurements of up to 4 meters with a coverage area of $240°$. For data acquisition, the Raspberry Pi (a single board computer) is used. The map where the real experiments are conducted is shown in Fig. 8a. The area of this map is approximately $49\ m^2$. Figure 8b shows the occupancy gird obtained after exploring the real map.

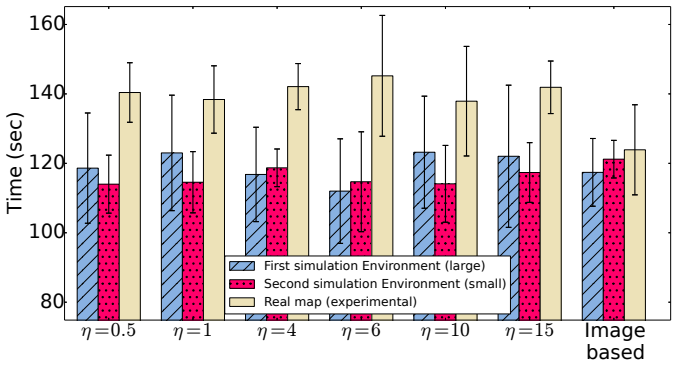
(a)



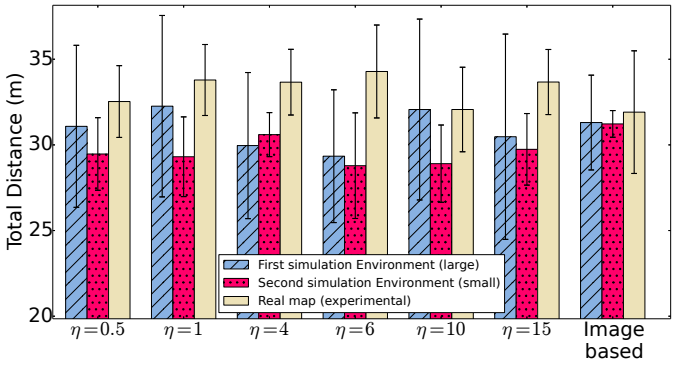
(b)

Fig. 8: The real map

In (a) is the real map used in the experiments. In (b) is the occupancy grid generated for the real map, it is also showing the robot after exploration has finished.



(a)



(b)

Fig. 9: Simulation and experimental results

VI. Results

As described at the beginning of Section V, 70 exploration runs are performed for 3 maps (2 simulation, 1 real). At the end of each exploration run, the total time taken for exploration and the total distance covered by the robot during exploration are recorded. All simulation and experimental results are shown in Fig. 9.

A. Simulation Results

The first set of results is for the first environment with the large map, shown in Fig. 7a, where the long range laser scanner is used.

The second set of simulation results is for the second environment with a small map, shown in Fig. 7b, where a low range laser scanner is used.

Simulation results show that using RRT-based detection does not compromise the efficiency of exploration, in terms of time and total distance needed to cover the map. The effect of laser scanner range is also insignificant, although it affects the speed at which RRT expands in the space. The effect of laser scanner range is compensated by the multiple RRTs (global frontier detector which never resets, and the local frontier detector which resets frequently).

B. Experimental Results

The last set of results is produced using the real experimental setup. The results agree in general with the above mentioned simulation results. The only difference is that, RRT-based experimental exploration is slightly more time consuming than the image processing based exploration, however we believe that such small differences in performance show that the proposed exploration strategy will be viable for 3D exploration, where image processing-based exploration techniques maybe unusable.

VII. Conclusion

In this paper, a new map exploration strategy is presented. The strategy is based on the RRT algorithm, where RRT is used to find frontier regions. Usual implementations of frontier detectors utilize image processing tools to extract frontier regions, this limits their application to 2D exploration. The proposed strategy uses RRT to detect frontier points. RRT is not limited to 2D space, hence, RRT can be applied to find frontier points in 3D map representations, which can in future allow for efficient 3D exploration. The proposed strategy is tested by performing a total of 210 exploration runs. Results obtained show that the proposed strategy can successfully extract frontiers and explore the entire map in a reasonable amount of time and cost, and without substantially losing performance when compared against image processing-based frontier detection techniques. Another contribution of this work is that, a custom ROS package for RRT-based exploration has been developed, and it is available for users at [23]. Future work will consider multi-robot 3D exploration, and possible different initiation points for the reset-able local trees.

References

[1] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '97)*. Washington, DC, USA: IEEE Computer Society, July 1997, pp. 146–151.

[2] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Proceedings of the Second International Conference on Autonomous Agents (AGENTS '98)*. New York, NY, USA: ACM, 1998, pp. 47–53.

[3] Y. Wang, A. Liang, and H. Guan, "Frontier-based multi-robot map exploration using particle swarm optimization," in *Proceedings of the IEEE Symposium on Swarm Intelligence (SIS)*, April 2011, pp. 1–6.

[4] P. G. C. N. Senarathne, D. Wang, Z. Wang, and Q. Chen, "Efficient frontier detection and management for robot exploration," in *Cyber Technology in Automation, Control and Intelligent Systems (CYBER), 2013 IEEE 3rd Annual International Conference on*, May 2013, pp. 114–119.

[5] M. Keidar and G. A. Kaminka, "Robot exploration with fast frontier detection: Theory and experiments," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '12, vol. 1. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 113–120. [Online]. Available: http://dl.acm.org/citation.cfm?id=2343576.2343592

[6] G. Oriolo, M. Vendittelli, L. Freda, and G. Troso, "The SRT method: randomized strategies for exploration," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA '04)*, vol. 5, April 2004, pp. 4688–4694.

[7] L. Freda and G. Oriolo, "Frontier-based probabilistic strategies for sensor-based exploration," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA '05)*, April 2005, pp. 3881–3887.

[8] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[9] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon "next-best-view" planner for 3d exploration," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1462–1468.

[10] H. El-Hussieny, S. Assal, and M. Abdellatif, "Improved backtracking algorithm for efficient sensor-based random tree exploration," in *Computational Intelligence, Communication Systems and Networks (CICSyN), 2013 Fifth International Conference on*, June 2013, pp. 19–24.

[11] A. Franchi, L. Freda, G. Oriolo, and M. Vendittelli, "The sensor-based random graph method for cooperative robot exploration," *IEEE/ASME Transactions on Mechatronics*, vol. 14, no. 2, pp. 163–175, April 2009.

[12] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based slam with Rao-Blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, pp. 2432–2437.

[13] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, Feb 2007.

[14] S. Mukhopadhyay and F. Zhang, "A path planning approach to compute the smallest robust forward invariant sets," in *In proceedings of the American Control Conference*, June 2014, pp. 1845–1850.

[15] P. Varnell, S. Mukhopadhyay, and F. Zhang, "Discretized boundary methods for computing smallest forward invariant sets," in *In proceedings of the 55th IEEE Conference on Decision and Control (CDC)*, Dec 2016, pp. 6518–6524.

[16] S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning," *ArXiv e-prints*, May 2010.

[17] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

[19] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at http://octomap.github.com.

[20] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: http://ijr.sagepub.com/content/30/7/846.abstract

[21] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, "Coordination for multi-robot exploration and mapping," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, Austin, TX, 2000, pp. 852–858.

[22] A. Romero. (2014, June) Ros concepts. Internet. [Online]. Available: http://wiki.ros.org/ROS/Concepts

[23] H. Umari. (2016, Sept.) RRT exploration ROS package. Internet. [Online]. Available: https://github.com/hasauino/rrt_exploration

[24] Gazebo simulator. Internet. [Online]. Available: http://gazebosim.org/

[25] Hokuyo URG-04LX laser scanner specifications. Internet. [Online]. Available: http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/URG-04LX_spec_en.pdf