An Automated Statistical Evaluation Framework of Rapidly-Exploring Random Tree Frontier Detector for Indoor Space Exploration

Wen-Chung (Andy) Cheng, Wen-Yu (Marty) Cheng, Zhen Ni, Xiangnan Zhong
Department of Electrical Engineering and Computer Science
Florida Atlantic University
Boca Raton, United States
{wcheng3, wcheng2014, zhenni, xzhong}@fau.edu

2022 4th International Conference on Control and Robotics (ICCR) | 978-1-6654-8641-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCR55715.2022.10053918

***Abstract*—This paper focuses on the design of an automated statistical evaluation framework for mapping generation of *Rapidly-Exploring Random Tree (RRT)* frontier detectors. By evaluating the run time and distance traveled of the simulated Kobuki robot agent in a Gazebo environment, the designed framework can automatically evaluate the process on a user-defined Gazebo map for a large number of repeated simulations. We also expanded the experiment platform into customized maps with complex layouts and trial schemes. The key formulas and parameters are provided with different trial settings. During the development of this framework, we have added functions that allow the user to choose among the maps we have designed, and the initial positions of the simulated robots for each map at the beginning of each trial. We have also modified the modules developed by Umari et al. so that the RRT frontier detection process can be started automatically with pre-defined exploration area in place. Modules have also been added so that the run time and distance traveled by the simulated robot for each trial can be measured and saved to the respective CSV files for further statistical analysis. We have created additional procedures that ensure the consistency of each trial. The results show that our designed automated evaluation framework is reliable and suitable for use as a fully automated research platform for robot exploration.**

Keywords—Frontier detection, rapidly-exploring random tree (RRT), simultaneous localization and mapping (SLAM), path planning, and statistical analysis

I. Introduction

Frontier-based exploration is one of the most common approaches for automated robot exploration. In frontier-based exploration, robots explore by repeatedly computing (and moving towards) frontiers, segments which separate the known regions from unknown [1]. Several algorithms have been designed for this application. In particular, two frontier-based exploration methods were initially developed: Wave-front Frontier Detector (WFD) and Fast Frontier Detector (FFD) [1]. Both methods do not need to process the entire map data, and it was shown that both methods have higher frontier point computation rates than the state-of-the-art method during that time by several orders of magnitude. However, both frontier exploration methods were not implemented for single or multi-robot SLAM map generation with simulated robot agents and environments. Thus, the average run time and total distance traveled by the agents while they explore and generate the whole map were difficult to obtain.

In 2017, Umari et al. developed a multi-rapidly-exploring randomized trees (RRT) method for autonomous robotic exploration [2]. The authors used multiple RRT's (global and local RRT) for seeking frontier points when generating a map of an unknown environment using SLAM. The purpose of the multiple RRT's is to increase the rate of frontier points generation. This method was tested in both simulated and physical environments to address its exploration efficiency as compared to the image-based method. In order to perform the comparative analysis between the proposed method and the image-based method in simulated environments, 70 exploration runs (10 for image-based method and 60 for proposed method) were performed on each map. However, the autonomous robotic exploration method developed by Umari et al. still required the manual design or control from the user in the process. For example, a user is required to define the area of exploration and start a new trial when the simulated robot finishes exploring a given map. This limitation makes the process time-consuming and would require constant user intervention if the user would like to dramatically increase the number of trials to gain further statistical insight of the frontier detector performance. Therefore, this paper aims to develop a more automated framework for evaluating the performance of future novel frontier detection methods.

The frontier-based approach, which is the most common approach for robot exploration, was first proposed by Yamauchi [3]. The frontier is a set of points that make up the boundary between the open space and the unexplored space [4]. Such approach has been extended to multiple robots, and image processing based method has been used for edge detection in multiple frontier based approaches [5]. However, as the dimension of the area of exploration increases, the image processing based approach becomes time and resource consuming.

One of the frontier-based methods, Rapidly-exploring Random Trees (RRT) [6] [7], is a fast probability path planning method with the ability to operate in 3-D space [8]. More

specifically, the RRT is probabilistically complete [9]. Thus this approach guarantees the complete discovery and the exploration of the whole environment. Using the RRT algorithm, an optimal sub-path can be generated from the starting position to the target position [10]. Umari [2] proposed the utilization of multiple RRT trees to detect frontier points. Moreover, since the RRT algorithm is biased towards the unknown regions, it can quickly detect the frontier points on the map [2].

On the other hand, Kontoudis and Vamvoudakis [11] introduced an online kinodynamic motion planning algorith- mic framework using asymptotically optimal rapidly-exploring random tree and continuous-time Q-learning (RRT-Q*). This approach offers an online optimal policy with asymptotic convergence properties and with completely unknown physics of the system. More specifically, they utilize integral reinforcement learning and a model-free Q-based advantage function to generate tuning laws for the online approximation of the optimal cost and the optimal policy of continuous-time linear systems. In addition, they propose a local static obstacle augmentation and a local re-planning framework to guarantee safe kinodynamic motion planning. However, RRT-Q* cannot be employed for unknown continuous-time non-linearizable systems. Moreover, this approach is not robust to disturbances such as external disturbances or measurement noise. This approach also employs static obstacle augmentations, which means the shape of the obstacles increases through time depending on the kinodynamic distance if the environment is dynamic.

The online resource of *rrt exploration* is a repository of single/multi-robot RRT map exploration algorithm for mo- bile robots that was developed for Robotic Operating Sys- tem (ROS). It uses occupancy grids as a map represen- tation [12]. Whereas *rrt exploration tutorials* is a comple- mentary package for the RRT Exploration package. The *rrt exploration tutorials* package was developed for ROS, and it provides all the needed Gazebo simulation files to bring up simulated Kobuki robots equipped with laser scanners and its physical properties [13]. However, the simulation process requires a user to launch the simulated environment program and the frontier detection program separately, and then manually define the area of exploration on the map for all of the trials. And when the current trial is done, the user has to manually start another trial using the previous steps. Performing the above tasks for any arbitrary number of trials to gain a statistical insight of the frontier detector performance can be time-consuming and would require constant user interventions to start new trials.

With the aforementioned observation, this paper expands the capacity of the existing RRT exploration packages [12] [13] by extending the program modules to include automated simulation, customized maps, performance evaluation, and so on. In specific, we have modified the RRT frontier detectors module from [12] so that it will automatically shutdown when the SLAM map of the environment is complete. Moreover, the statistical results of the evaluation, such as run time and the distance traveled by the simulated robot during the trial, can now be saved as designated CSV files at the end of each trial for further analysis. A few customized maps are created as well to show the robustness/scalability of the framework.

The organization of the paper is provided as follows. The traditional implementation by Umari et al. is presented in section II. The automated framework development is presented in section III. The problem set-up along with the set parameters for each environment is presented in section IV. The results with relevant figures are provided in section V. Finally, section VI concludes the work.

II. Background and Traditional Implementation

As previously stated, RRT-based frontier detector modules discover frontier points. 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 [2]. 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. Note that 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. Umari et al. proposed two versions of frontier detectors: i) a local frontier detector; and ii) a global frontier detector [2].

A. Local RRT

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 $x_{rand} \subset X_{free}$, the free space is sampled using the SAMPLEFREE function. Then, the STEER function 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 $E = \phi$). 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 is one of the major differences between the usage of RRT for exploration in [2], compared to other standard implementations of RRT available in literature. 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. The local detector is proposed by Umari et al. for fast detection of frontier points in the immediate vicinity of the robot at any time [2].

B. Global RRT

The implementation of the global frontier detector is identical to that of the local frontier detector, except that the

global 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. The authors in [2] proposed to use global and local RRT detectors because:

- This allows detection of frontier points quicker due to the local detector tree always starts growing from the robot's current position, which increases the chance that the next point picked from the RRT for exploration lies in the unknown space.
- 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, the global frontier detector is used.

C. Strategy Description

The exploration strategy proposed by [2] 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. 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. For an overall high level schematic diagram of the exploration strategy, please refer to Figure 1 from [2]. Thus, the task allocator module will need to terminate the whole exploration algorithm when it receives no more clustered frontier points from the filter module in order for the trial to end automatically once the SLAM map of the environment is generated completely. This will be explained in greater detail in section III-A.

D. ROS Implementation

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. For more details on the ROS implementation of the exploration strategy described above, please refer to Figure 6 from [2]. Different pre-built ROS packages are used in the implemen- tation for mapping and path planning. Also the 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 [14] [15]. The ROS Navigation stack is used to control and direct the robot towards exploration goals (i.e. the assigned clustered frontier points). Path planning based on the A* algorithm [16] is one of the packages already available within the ROS navigation stack, which is used 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, 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. For details related to ROS terminology (publish, subscribe, etc.) please refer to [17], and for specific details related to the actual implementation by [2] please see [12].

III. Proposed Framework Development

In this section we will describe the design of the automated framework for evaluating performance of RRT frontier detection.

A. Automated Framework Development

At the beginning of each exploration trial, the exploration area of the unknown environment needs to be defined by the manually clicking corner points of the area (upper, lower right and left), and the root point of the RRT exploration tree starting right from the middle of the area. Both the local and global frontier detector nodes described in section II-D subscribe to a common ROS topic called *"/clicked point"*, which publishes coordinates of clicked points on rviz visualizer (rviz is a 3D visualizer for the ROS framework [18]) made by the user. The coordinates of the clicked points are in meters.

To make the exploration defining task automatic, the clicked points on the rviz visualizer published to the local and global detector nodes were hard-coded to the following: (-16.82, 13.86), (-16.23, -13.81), (15.70, -13.92), (16.26, 12.31), (0, 0); where coordinates (0, 0) is the coordinates of the root point of the local and global RRT exploration trees, which is usually the center of the map. These coordinates that define the exploration area covers most of the environments described in section IV-A.

To make the RRT exploration terminate automatically when the SLAM map of a given environment is complete, the robot task allocator described in section II-D needs to be modified. Since the robot task allocator was written as a while loop that terminates only when a user shuts down the whole process manually, and the robot task allocator receives new frontier points by subscribing to the ROS topic called *"/filtered points"* which is published by the filter node, a new condition that terminates the whole exploration when no new frontier points are received by the allocator was added.

While developing the RRT frontier detector evaluation framework, the Gazdebo simulator crashed for some trials. Therefore a "restart trial" condition for the Gazebo program was added to the framework. Figure 1 shows the overall RRT frontier detector performance evaluation structure for each trial. For each trial, the environment name and the number

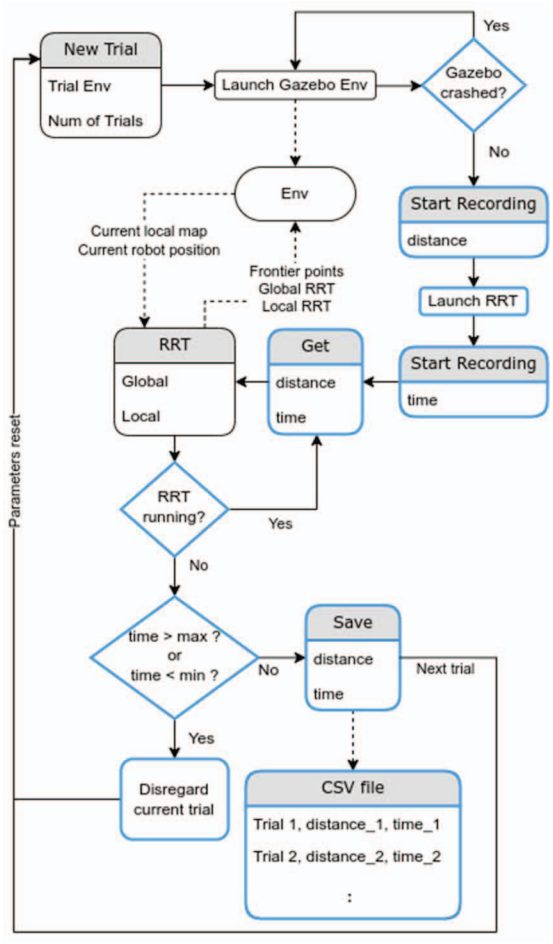

Fig. 1. RRT frontier detector evaluation framework structure for each trial. The processes boxed in blue are the modifications made by us. The RRT frontier detection process has been modified so that publish points that designate the exploration area can be automatically assigned at the start of each trial. Auxiliary processes such as Gazebo simulator crashing preventor have been added to ensure the consistency of each trial.

of trials are provided as inputs. A process will then launch the Gazebo simulator with the environment associated with the given name. If in the process the Gazebo simulator crashes, the current trial will be restarted. Next the distance traveled by the simulated robot will be initialized to 0 meters since the robot has not moved yet. Then the RRT frontier detection process will be started. At this stage, a timer will be started to measure the elapsed time of the exploration of whole environment while mapping. For each iteration of RRT frontier detection, the distance traveled by the simulated robot and the time elapsed will be measured. Both the global and local RRT frontier detectors take in the current local map and the robot position as inputs, and then output the new frontier points, global and local RRT tree edges to the Gazebo environment. A process will then check if the RRT frontier detection process is still running. If the RRT frontier detection process is terminated, the elapsed time of the process will be checked if it is larger than the maximum cutoff time or less than the minimum cutoff time. The purpose of the maximum and minimum cutoff time is to eliminate trials where the simulated robot hangs at one place for a substantial amount of time, or trials where the RRT frontier detection process terminates before the whole SLAM map of the given environment is generated. If the elapsed time of the process satisfies the above conditions, the current trial will be restarted. Otherwise, the time elapsed of the process, the distance traveled by the simulated robot, and the generated SLAM map of the given environment, depending on the user's choice on saving the map, of the current trial will be saved. The next trial is then continued as the above routine.

B. Automatic RRT Program Design

To start the exploration trial without launching separate processes by hand as described in section I, a whole new workflow is constructed. Below are the procedures used in the workflow:

- SPAWNROBOTINENV$_i$(x_{Pos_i}, y_{Pos_i}, z_{Pos_i}): Spawns a simulated robot at location (x_{Pos_i}, y_{Pos_i}, z_{Pos_i}) of a given environment Env_i.
- INITMOVEMENTDETECTOR(): This is a custom class constructor for measuring total distance traveled by simulated robot in the environment. Its "distance" attribute saves the updated total distance traveled by the simulated robot in the given environment.
- LAUNCHRRTFRONTIERDETECTOR(): This is a custom function for launching the RRT frontier detector process.
- RECORDCURRENTTIME(): This is a built-in Python function that displays current time in UTC.
- RRTFRONTIERDETECTORRUNNING(): This is a custom function that returns TRUE if the RRT frontier detector process is running.
- OUTPUTGENERATEDMAP(): This is a built-in ROS function that saves the resulting SLAM map of the environment.
- DISTANCEADDTOCSV(d_{trial}): This is a custom function that saves the distance traveled by the simulated robot during the current trial, d_{trial}, to dedicated CSV file.
- TIMEADDTOCSV(t_{trial}): This is a custom function that saves the run time of exploring the whole environment during the current trial, t_{trial}, to dedicated CSV file.

The automation workflow is split into 3 processes to reduce the complexity of the individual task:

- Environment Selector: This procedure accepts the following as inputs:
  - *Name of the environment (Env)*: the name of the environments is entered as a string. The available environments for this workflow were described in section IV-A.
  - *Initial position of robot (Pos)*: the initial position of the simulated robot is entered as a string ("C" for Center, "UR" for upper right, "LR" for lower right, "LL" for lower left, and "UL" for upper left). The available initial positions are already pre-defined for each environment described in IV-A.
  - *Number of trials (Num)*: the number of trials to perform the RRT exploration on the given environment

is entered as an integer. The experiment will run for as many trials as indicated by this number.

– *Save map option (Map)*: the option to save the generated SLAM will be entered as a Boolean variable. When enabled, the generated SLAM map will be saved to local drive at the end of each trial. This can be used to judge if the completed trial was successful.

This procedure processes the environment choice entered by the user. There is an RRT exploration trial process for each included environment, so this procedure can be expanded to any number of environments. For each environment, the remaining user inputs are passed to the next process, which is the initial position selector.

- Initial Position Selector: There can be as many initial position selectors as the number of available environments. This procedure accepts the unprocessed inputs passed by the environment selector, and processes the initial position choice entered by the user. There is an RRT exploration trial process for each initial position, so this procedure can be expanded to any number of positions. For each position, the remaining user inputs are passed to the next process, which is the core process of the RRT exploration automation trial.
- RRT Starter: This procedure accepts the unprocessed inputs passed by the initial position selector, and processes the number of trials to be run and option to save generated SLAM map entered by the user. The process iterates through each trial, so this can be ex- panded to any number of trials. For each trial, the proce- dure SPAWNROBOTINENV$_i$(x_{Pos_i}, y_{Pos_i}, z_{Pos_i}) spawns the simulated robot into the i^{th} environment at the i^{th} initial position with coordinates (x_{Pos_i}, y_{Pos_i}, z_{Pos_i}). Then a movement detector is initialized using the INITMOVEMENTDETECTOR() procedure so that the distance traveled by the simulated robot is measured at all times. LAUNCHRRTFRONTIERDETECTOR() then launches the RRT frontier detection process for the robot. RECORDCURRENTTIME() then records the starting time to a variable named t_{start} While the RRT frontier detection process is running, the running time $t_{running}$ is recorded by RECORDCURRENTTIME(). The updated elapsed time $t_{elapsed}$ is then calculated by finding the difference between $t_{running}$ and t_{start}. The up- dated distance traveled by the robot is then recorded using the ”distance” attribute of the initialized move- ment detector. If at any moment the elapsed time $t_{elapsed}$ exceeds the maximum cutoff time t_{max}, the RRT frontier detector process will be terminated by setting RRTFRONTIERDETECTORRUNNING() to FALSE.
When the RRT frontier detection process terminates, the elapsed time $t_{elapsed}$ will be checked if it's greater than the maximum cutoff time t_{max} or less than the minimum cutoff time t_{min}. If the above condition is met, the current trial will be ignored. Otherwise, both the elapsed time $t_{elapsed}$ and distance traveled $d_{traveled}$ by the robot will be recorded to dedicated CSV files for later analysis using DISTANCEADDTOCSV(d_{trial}) and TIMEADDTOCSV(t_{trial}). The next trial is then proceeded with the above routine.

IV. EXPERIMENTS SET-UP

A. Environments Used in Simulations

The two environments used for this framework are created using the Gazebo simulator [19], which provides realistic robotic movements, a physics engine, and the generation of sensor data combined with noise.

a) Apartment: The first environment shown in Fig. 2, is a customized map with an area of approximately 82.75 m^2 (free space area).

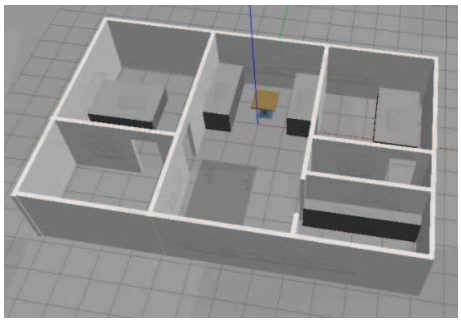

Fig. 2. The custom apartment environment. Dimension = 12.50 m×8.00 m. Environment size ≈ 82.75 m^2.

b) Office: The second environment shown in Fig. 3, is a customized map with an area of approximately 267 m^2 (free space area).

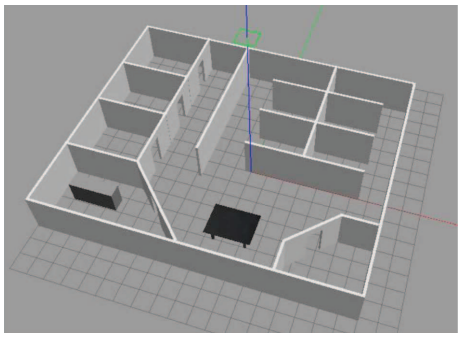

Fig. 3. The custom office environment. Dimension = 18.00 m × 15.00 m. Environment size ≈ 267 m^2.

B. Parameters Used in Simulations

a) Apartment Environment: The following are the parameters used for this particular environment:

- Minimum cutoff time: 80 seconds
- Maximum cutoff time: 300 seconds

These parameters are set since the RRT exploration trial can sometimes prematurely end due to SLAM map refresh rate and frontier points update update mismatch.

b) Office Environment: The following are the parameters used for this particular environment:

- Minimum cutoff time: 80 seconds
- Maximum cutoff time: 300 seconds

These parameters are set since the RRT exploration trial can sometimes prematurely end due to SLAM map refresh rate and frontier points update update mismatch.

V. Results and Analysis

A. Apartment Environment

Figures 4 and 5 show the histograms of distance traveled by the robot and elapsed time over 100 trials when the robot starts from the center initial position.

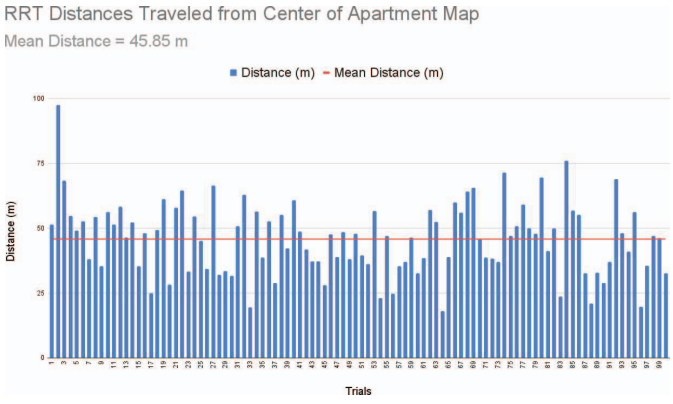

Fig. 4. Distance traveled histogram for center initial position at the Apartment environment.

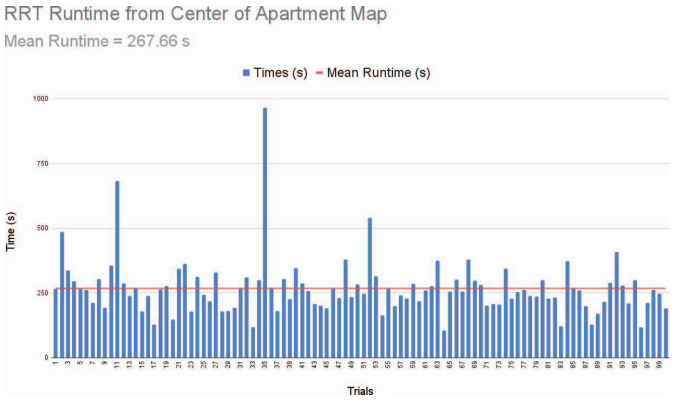

Fig. 5. Time elapsed histogram for center initial position at the Apartment environment.

We have conducted 100 trials with various initial positions. Figures 6 and 7 show the box plots of distance traveled and time elapsed for all the initial positions. As can be seen in the box plots, the center initial position yields the longest median distance traveled. This, as can be observed from Figure 2, is due to the robot starting from a region with obstacles (tables) in that room. The robot would travel back and forth more often to find the assigned frontier points around those obstacles. Whereas the lower right initial position yields the longest median time elapsed. This, again can be observed from Figure 2, is due to the robot starting from a narrow region. The robot would sometimes hang in place thus this initial position does not yield longer distance traveled than the center.

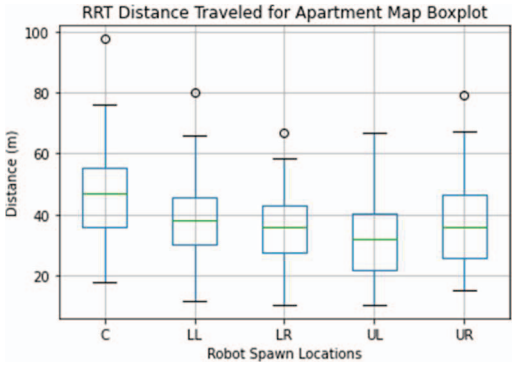

Fig. 6. Distance traveled box plot for the Apartment environment.

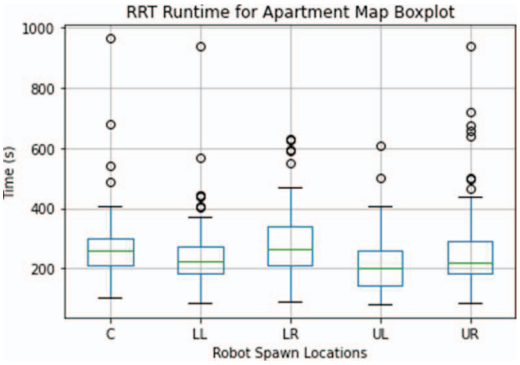

Fig. 7. Time elapsed box plot for the Apartment environment.

B. Office Environment

We have also conducted 100 trials with various initial positions for this environment. Figures 8 and 9 show the box plots of distance traveled and time elapsed for all the initial positions. As can be seen in the box plots, the lower right initial position yields the longest median distance traveled and time elapsed. This, as can be observed from Figure 3, is due to the robot starting from a more restricted region by the surrounding walls in that room. The robot would travel back and forth more often to find the assigned frontier points, and the RRT frontier detector tree would need more time to expand to the other rooms. Whereas the upper left initial position yields the shortest median distance traveled and time elapsed. This, again can be observed from Figure 3, is due to the robot starting from a more spacious region. The robot would travel in straight line to find the assigned frontier points most of the time, and the RRT frontier detector tree would expand to the other rooms faster.

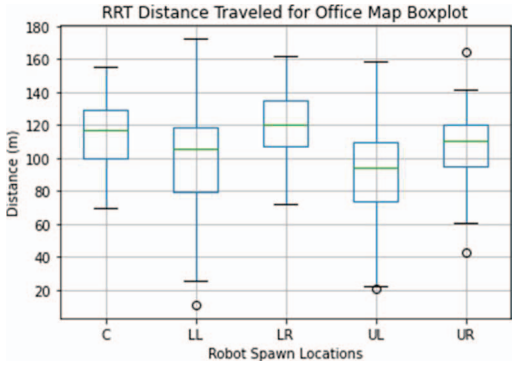

Fig. 8. Distance traveled box plot for the Office environment.

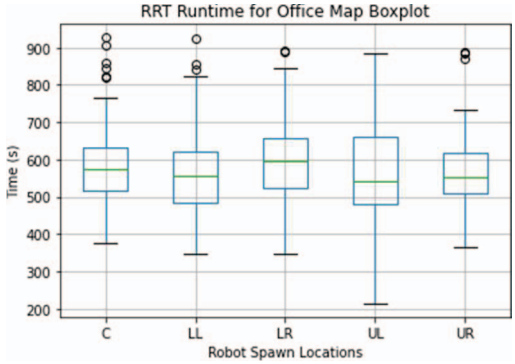

Fig. 9. Time elapsed box plot for the Office environment.

VI. Conclusions

In this work, we developed an automated RRT frontier detection evaluation framework based on the module developed by [2] that is implemented in ROS. The evaluation framework is able to automate any desired number of trials using RRT without constant user intervention, including the ability to start and end each trial automatically, as well as collect statistical results that can be saved in CSV format for further external analysis. Furthermore, the framework can also be configured with custom map layouts and trial parameters, such as initial positions, allowing for high levels of customization that can be tailored to any future research needs.

Acknowledgement

This work was partially supported by the National Science Foundation under Grant 1947418.

References

[1] 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-Volume 1*, 2012, pp. 113–120.

[2] H. Umari and S. Mukhopadhyay, "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1396–1402.

[3] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*. IEEE, 1997, pp. 146–151.

[4] Z. Yan, L. Fabresse, J. Laval, and N. Bouraqadi, "Metrics for performance benchmarking of multi-robot exploration," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 3407–3414.

[5] W. Qiao, Z. Fang, and B. Si, "Sample-based frontier detection for autonomous robot exploration," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2018, pp. 1165–1170.

[6] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[7] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[8] L. Zhang, Z. Lin, J. Wang, and B. He, "Rapidly-exploring random trees multi-robot map exploration under optimization framework," *Robotics and Autonomous Systems*, vol. 131, p. 103565, 2020.

[9] 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.

[10] A. Ivanov and M. Campbell, "An efficient robotic exploration planner with probabilistic guarantees," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 4215–4221.

[11] G. P. Kontoudis and K. G. Vamvoudakis, "Kinodynamic motion planning with continuous-time q-learning: An online, model-free, and safe navigation framework," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 12, pp. 3803–3817, 2019.

[12] H. Umari, "rrt exploration," https://github.com/hasauino/rrt exploration, 2016.

[13] ——, "rrt exploration tutorials," https://github.com/hasauino/rrt exploration tutorials, 2016.

[14] 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*. IEEE, 2005, pp. 2432–2437.

[15] 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, 2007.

[16] 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, 1968.

[17] A. Romero, "Ros concepts," http://wiki.ros.org/ROS/Concepts, 2014, [Online; accessed 27-Apr-2022].

[18] Open Source Robotics Foundation, "rviz," https://github.com/ros-visualization/rviz, 2012.

[19] Gazebo, "Open Source Robotics Foundation," http://gazebosim.org/, 2014, [Online; accessed 24-Apr-2022].