Multi-Robot Exploration of Unknown Indoor Areas

Submitted by

Yep Yan Ling

A0205736E

Department of

Mechanical Engineering

In partial fulfilment of the

requirements for the Degree of

Bachelor of Engineering

National University of Singapore

AY2021/2022

25 March 2022

SUMMARY

Coordination of a multi-robot team for exploration is increasingly favoured over a single robot for its associated efficiency and robustness. To fulfil different objectives during explorations, such as search and rescue or map construction, the coordination strategies employed must also be adapted. For that, this project aims to develop a programme to support the testing of coordination strategies in multi-robot explorations for a fast and fair evaluation of diverse approaches. There are three components for this test-simulator—first, a multi-robot simulator for the unique representation of 2D environments. Second, an exploration module for baseline comparison. Third, consistent measurements and evaluation of the performance of exploration run.

The programme is built on the Robot Operating System (ROS) to leverage existing ROS packages for simulation. The exploration module, Coordinated Frontier-Based Exploration (CFE), was written in C++ and pursued frontiers with the highest utility. In addition, scripts were created to collect and analyse various performance metrics automatically. Three ROS packages have been implemented to fulfil the functions.

CFE has coordinated the multi-robot team to explore four proposed environments efficiently. Its metrics were measured and validated against an open-source code based on Rapidly-exploring Random Trees (RRTs). CFE was found to perform better in most environments, attributing to its ability to distribute the robots and evaluate the feasibility of centroids.

To realise the benefits of multi-robot systems, future works include building a navigation module that takes in the produced merged map and returns a globally optimal path. In addition, other frontier detection methods can be tested for the extension to larger environments.

ACKNOWLEDGEMENT

I would like to extend my deepest gratitude to my Project Supervisor, A/Prof Guillaume Sartoretti, for his support and guidance. His insightful feedback and belief in me have pushed me to sharpen my thinking and brought my work to a higher level. I would also like to thank the Engineering Scholars Programme for the support and opportunities it offered throughout my studies. Lastly, I would like to thank Lingshan for her unwavering encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

All symbols have been defined where they were listed for easier reference.

1. INTRODUCTION

Robotic exploration is the strategic navigation of robots to efficiently gain information about a typically unknown environment and has seen critical applications in rescue, planetary exploration, and cleaning [1, 2]. Multi-robot exploration boasts obvious benefits over single robot exploration, notably faster exploration and greater fault tolerance [3]. As summarised by [4, 5], the challenges of multi-robot exploration include coordination, map merging, and limited communication of the robot team, with coordination being the focus of this paper as it strongly affects the quality of exploration [6].

Coordination involves a centralised or decentralised planner that maximises exploration efforts by distributing tasks within the robot team [7]. It is accompanied by improved performance metrics like exploration time, cost, safety, map completeness, and quality [8]. Coordination is comprised of two aspects, identification of exploration targets and allocation of robots to the targets.

One main exploration method is the frontier-based exploration. Built on the concept of frontier, the boundary between known and unknown areas as established in [9], other works have based various coordination strategies on exploring these frontiers. Coordinated Multi-Robot Exploration (CME) optimises an objective function, also known as utility, consisting of cost and information gain for efficient explorations [1, 4, 5]. However, since the algorithm greedily maximises utility at individual time steps without considering the impact on future selections, it arrives at sub-optimal solutions in environments where less optimal frontiers must be pursued momentarily for more efficient exploration in the future. An example would be corridors, which often have low utility. However, the pursuit of it speeds up exploration due to its branching into unknown regions [10].

Other exploration strategies have identified other regions of interest to better estimate the potential impacts of the targets on exploration. Semantic information is used to determine the spatial identity and direct exploration efforts to areas of higher interest (e.g., corridors) [10, 11]. Rapidly-exploring randomised Trees (RRTs) are used to bias search towards unexplored areas [12]. These methods improve the quality of targets used for exploration by encapsulating higher-level information.

The interest in multi-robot exploration will continue to expand, specifically in complex and uncertain environments. A large proportion of exploration algorithms extend the framework of CME in [1], incorporating mapping quality [3, 4] or communication constraints [2, 13, 14] to tackle other challenges of exploration. As such, simulators for multi-robot explorations are important for fast and reliable testing of the growing exploration strategies [8].

While CME theories are well-established, the number of open-source platforms for the testing of the exploration theories, which includes validation and benchmarking, is lacking. For that, this project aims to develop a multi-robot exploration test-simulator for a 2D indoor environment on the Robot Operating System (ROS) [15]. Extending the works of [8], the test-simulator will possess simulation capabilities (i.e., simulate desired number of robots in desired environments) and performance evaluation capabilities (i.e., record the robots’ status and compute respective metrics). An exploration module based on CME will also be included to benchmark against any proposed exploration strategies. The completion of this project will produce a multi-robot test-simulator for quick testing of exploration strategies, with documentations for reproducibility. This paper is organised as follows: Section 2 provides a review on related exploration simulators and strategies. Section 3 proposes the project approach. Section 4 outlines the implementation. Section 5 presents the results. Section 6 discusses the reasons for results. Section 7 concludes and provides recommendations for future works.

2. RELATED WORKS

The architecture for multi-robot testbed, along with an open source implementation, can be found in [8]. The paper provided guidelines on improving reproducibility of experiments through specifying experimental designs like stop conditions and metrics. While the open-source code supports simulations of robot teams in different environments using the Modular OpenRobots Simulation Engine (MORSE) [16], two areas could be further improved. Firstly, the proposed exploration module [17] did not employ any coordination strategy during the exploration, making it unsuitable as a baseline for comparison of coordination strategies. Secondly, the test-bed did not provide a measurement of its proposed metrics like exploration time, which increases the difficulty in quantifying performance.

rrt_exploration is an open-source multi-robot exploration strategy developed by [12]. The randomness associated with RRTs allow frontier points to be generated in unexplored areas, and tasks allocations is achieved using a market-based strategy. Like CME, every robot submits bids for the tasks based on individual costs and information gain. Rather than choosing the task with the highest revenue as in CME, an assigner allocates tasks to the highest bidders. The effect of task allocation is expected to be similar to CME, though using a centralised assigner may create a single point of failure. Overall, *rrt_explora*tion package (RRT) would serve well compared to the base CME module that would be implemented.

3. PROJECT APPROACH

The project has three areas of focus, a simulator, an exploration module, and performance evaluation, which have to be solved in the listed order. In this section, the architecture for the simulation, including the expected inputs and outputs of the exploration module, will be explained. Then, the method to measure the proposed metrics for performance evaluation will be presented.

3.1. Proposed Simulation Setup

The simulator has to first simulate different indoor environments and then spawn multiple robots in the specified environment.

3.1.1. Indoor Simulation Environments

As described in [8, 18], an indoor environment is generally flat and surrounded by walls. It can be categorised into office, maze, underground base. A warehouse is included due to its potential in exploration problems. The proposed environments are found in Figure 1.

- Blocks: High density of junctions and obstacles
- Maze: Most complex environments with dead ends
- Office: Separated areas of interest in the form of rooms
- Warehouse: Ordered obstacles for long straight routes

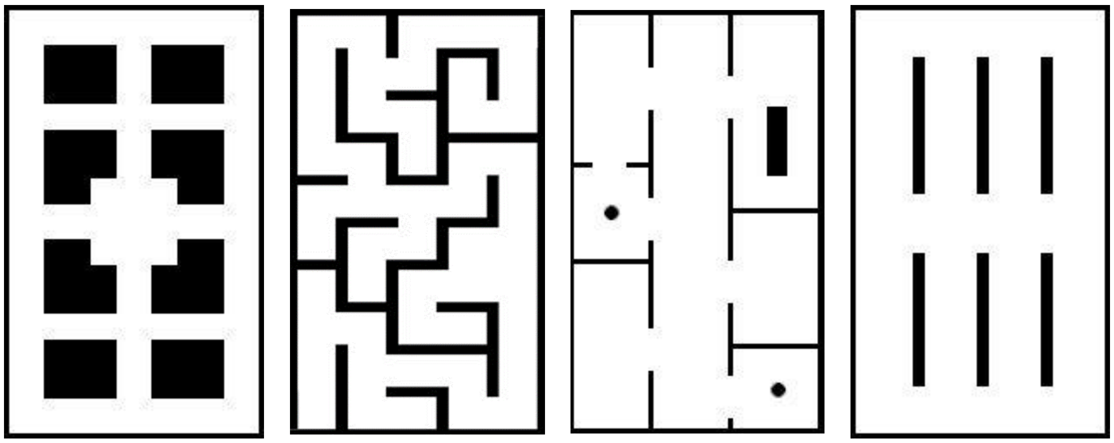

(a) Block (b) Maze (c) Office (d) Warehouse

Figure 1: Proposed Indoor Environments

Gazebo was chosen as the physical simulator for its realistic robotic movement and sensor, compatibility with ROS, support for multi-robot simulation, and ability to simulate different robot models [19]. The environments were generated with two ROS packages, *map_server* and *map2gazebo* (see Appendix A for instructions), which converted the initial image files to meshes for Gazebo. For easy adjustments of robots' starting positions, shell scripts were used to set the robots' poses, which include their x and y coordinates, and their orientation. Two initial configurations, which spawn the robots at positions far from each other and near each other, titled *far* and *near*, have been provided for each environment. This is to allow more robust testing of exploration strategies. The initial configurations can be found in Appendix B.

3.1.2. Multi-Robot Simulator

The task of creating the Multi-Robot Simulator is decomposed into simulating multiple single robots and subsequently, enabling communication between robots.

Figure 2 outlines the proposed architecture of the Multi-Robot Simulator (comprised of existing ROS packages) and its interaction with self-written Exploration Modules and Metrics Measurement.

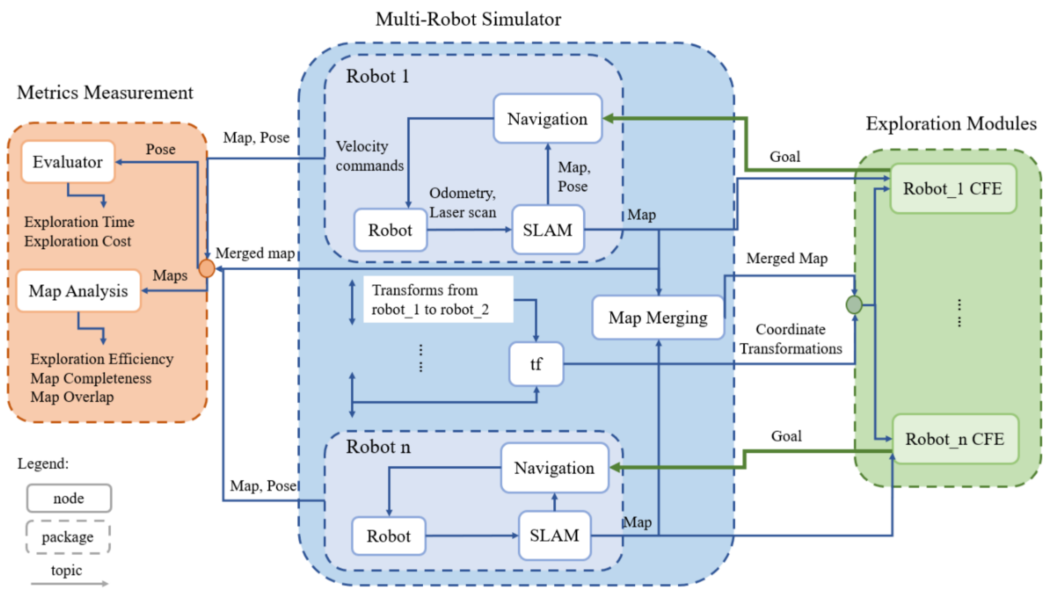


Figure 2: Proposed Software Architecture of the System

Every robot must possess the ability to localise, map, and navigate to waypoints as specified by the exploration module. For coordination, two ROS nodes will be added for frame transformations and map exchange. The functions and implementations of *SLAM*, *Navigation*, *tf*, and *Map Merging* nodes will be discussed.

Simultaneous Localisation and Mapping (SLAM) Node: This is required to produce an accurate global map with information on the robot's position in the map. Due to drift in odometry, there exist errors in robot pose estimation and subsequently in map construction. SLAM uses odometry and sensor information at different poses to compute the poses and corresponding sensor information that reduces the observed drift [20]. ROS gmapping package, which uses Rao-Blackwellized particle filters to match sensor readings, is selected due to its efficiency, popularity in robotic research, and compatibility with the other ROS packages [21].

Navigation Node: The navigation problem includes path planning, which is returning a collision-free minimum cost path [22], and motion planning, which is determining a set of motion controls to fulfil the desired trajectory [23]. The ROS *move_base package* from the ROS Navigation Stack uses global and local planners for path planning and motion planning, respectively, to compute a set of velocity commands for navigation to specific goals. The move base node then sends the velocity commands at intervals to the robot controller, which enables the robot to move.

Transform (tf) Node: The ROS tf package stores the relation between individual robot's coordinate frames (e.g., global, odometry and base). To differentiate the between robots, tf_prefix is uniquely set such that the robot's name will precede the common coordinate frames. Every robot can then obtain information about the other robots' poses if their initial coordinate transformations are specified. These transformations are obtained during the initialisation, as previously discussed.

Map Merging Node: Map merging enhances the accuracy of mapped regions and limits redundant exploration through the integration of multiple robots' explorations. Map merging techniques vary across map types, and there are four methods for occupancy grid maps: probability, optimisation, feature matching, or Hough transform [24]. The techniques utilise different aspects of the information grid to compute the most accurate transformation between robot maps. For this application, the ROS *multirobot_map_merge* package is initially selected for fast, reliable, and scalable merging for multi-robot explorations [25]. It utilises an Oriented FAST and rotated BRIEF (ORB) feature matching algorithm. Robots will exchange information on their occupancy grids via this package and obtain the merged maps. However, the maps generated on the fly suffered from great inaccuracies during simulations (see Figure 11(a)), and solutions were proposed in Section 4 to circumvent this issue.

3.2. Proposed Performance Metrics Measurement

Performance of the exploration is quantified with the robots' poses over time, the individual maps, and the merged maps. The proposed metrics common to state-of-the-art research are exploration time, exploration cost, exploration efficiency, and map completeness [8]. This list is non-exhaustive. An additional metric, map overlap, is proposed to measure the distribution of team. The two nodes, *evaluator* and *map analysis* (see Figure 2), measures metrics related to robot's pose and maps, respectively. Map overlap is calculated offline (see Appendix C). The evaluator node produces the trajectory of the robot under the */traj* topic for visualisation. The calculation for the proposed metrics is outlined below:

Exploration time: The time taken to complete (above some threshold) the exploration of the environment. The significance of this metric is straightforward.

Exploration cost: The distance, average or total, travelled by each robot during the exploration. This determines the redundancy in robots' paths and is often positively correlated with exploration time. The distance travelled by a robot is the summation of the Euclidean distances between its previous and current pose at fixed intervals.

Exploration efficiency: The ratio of explored area to cost incurred. It is interpreted as the amount of area of environment explored with every 1m travelled. This is particularly useful for applications where the exploration is incomplete, either due to a complex environment or a time-based terminating condition. By integrating two factors, it provides more conclusive evidence on the performance

Map completeness: The ratio of explored area in the merged map to the total explorable area in the ground truth map. This is also suitable for incomplete explorations. With the ground truth map (can be created in Section 3.1.1), the explorable area is the summation of known map cell with values greater than 0. The sum, when multiplied by the square of the map's resolution, gives area in m^2, which allows for the comparison of maps with differing resolutions.

Map overlap: The ratio of overlap area to the total map area for the merged map. An area is as an overlap if two of more robots possess information on it. Pairwise matching of robots' individual map is executed with two for loops to determine the total overlap areas in the merged map (see Appendix C for visualisation). The individual maps are transformed in the same frame as the merged map using the initial specified transformations. While feature detection and matching may be more robust, this method works for applications where initial positions are known.

4. IMPLEMENTATION

This section covers the theory and details of the implementation of the exploration module. The proposed architecture in Figure 3 highlights the key features: detector, filter, and allocator. The detector is in charge of frontier detection and clustering the centroid points for a given map. The filter takes in generated centroids and filters out invalid centroids to guide robots to unexplored regions. Lastly, the allocator evaluates the centroid points based on cost, information gain, and most importantly discount (computed from other robots' goals and pose). In case of communication loss, the robot may continue with the exploration based on centroids generated by its own detector, which is a benefit of this hybrid allocator system. The motivation behind using individual robot maps instead of the merged map arose due to observed inaccuracies of the merged map. The filter is then created to update the team's exploration efforts so that centroids are accurately generated in unexplored regions.

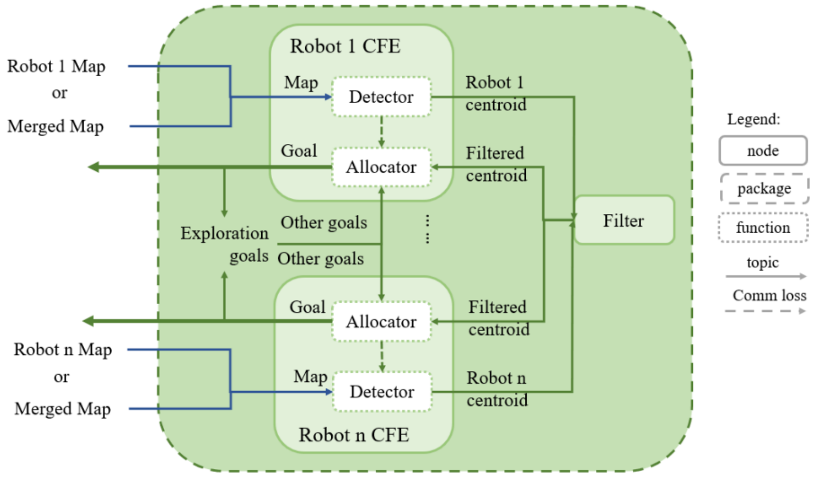
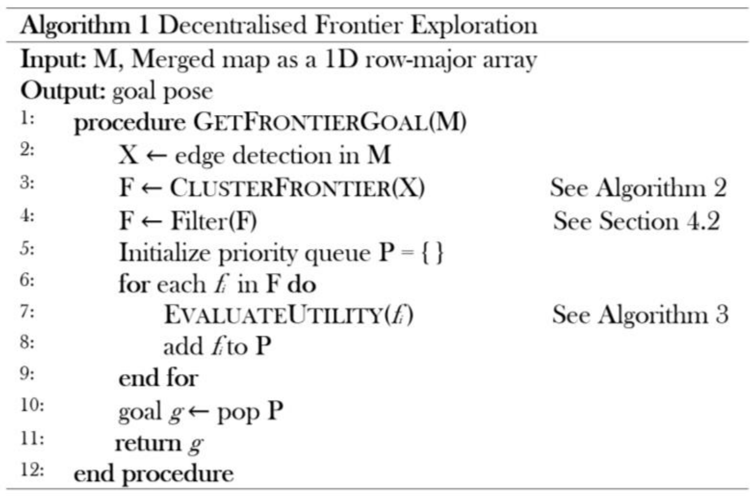


Figure 3: Architecture of Exploration Module

This exploration module extends the theory of CME in [1], with some variation due to the inclusion of frontier clustering and robot pose for utility calculation during frontier allocation. CFE is described by the detector, the filter, and the allocator. The pseudocode in Figure 4 outlines the code intuition for the three focuses. They are 1) Frontier Detection and Clustering (lines 2-3) and 2) Compilation of Frontier (line 4) and 3) Coordinated Frontier Allocation (lines 5-10).

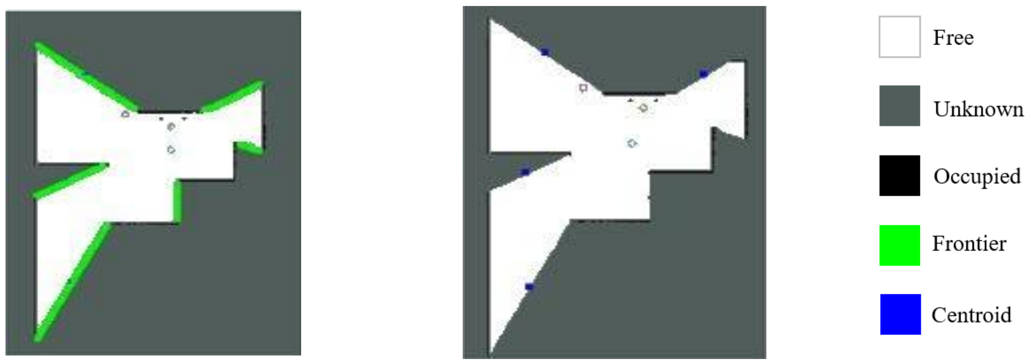


```
Algorithm 1 Decentralised Frontier Exploration
Input: M, Merged map as a 1D row-major array
Output: goal pose
1:  procedure GETFRONTIERGOAL(M)
2:      X ← edge detection in M
3:      F ← CLUSTERFRONTIER(X)                See Algorithm 2
4:      F ← Filter(F)                         See Section 4.2
5:      Initialize priority queue P = { }
6:      for each $f_i$ in F do
7:          EVALUATEUTILITY($f_i$)            See Algorithm 3
8:          add $f_i$ to P
9:      end for
10:     goal $g$ ← pop P
11:     return $g$
12: end procedure
```

Figure 4: Algorithm 1

4.1. The Detector: Frontier Detection and Clustering

The approach to this problem is analogous to solving the problem of edge detection and clustering in computer vision [9]. The occupancy grid is treated as an image with pixel values ranging from 0 to 100 for known regions and -1 for unknown regions. Frontier cells, the boundary between known and unknown regions, will be detected as edgels between the free and unknown cells in the occupancy grid. Edge detection is done by taking the first derivative of the grid cell to detect changes in cells' values, specifically using a Sobel filter. Then, the frontier cells are clustered to reduce dimension for future computation by exploiting that close frontier cells typically represent the same region.



(a) Frontier Detection (b) Frontier Clustering

Figure 5: Detection and Clustering of Frontiers

Clustering is done through a radius search heuristic (see Figure 6), where cells belong to the same cluster if they are within a certain radius, and the frontier centroid is returned as the mean of all points in the cluster.

```
Algorithm 2 Clustering Frontier Cells using Radius Search Heuristics
Input: X,  List of frontier cells coordinates
Output: F,  List of frontier centroids coordinates
1:  procedure CLUSTERFRONTIER(X)
2:      Initialize open list O = [first element of X]
3:      while X ≠ ∅ do
4:          Initialize list C = [ ]
5:          while O ≠ ∅ do
6:              $v$ ← pop O
7:              erase $v$ from X
8:              add $v$ to C
9:              N ← frontiers within specified radius of $v$
10:             add N to O
11:         end while
12:         if size of C > minimum cluster size then
13:             (i, j) ← mean(C)
14:             Add (i, j) to F
15:         end if
16:     end while
17:     return F
18: end procedure
```

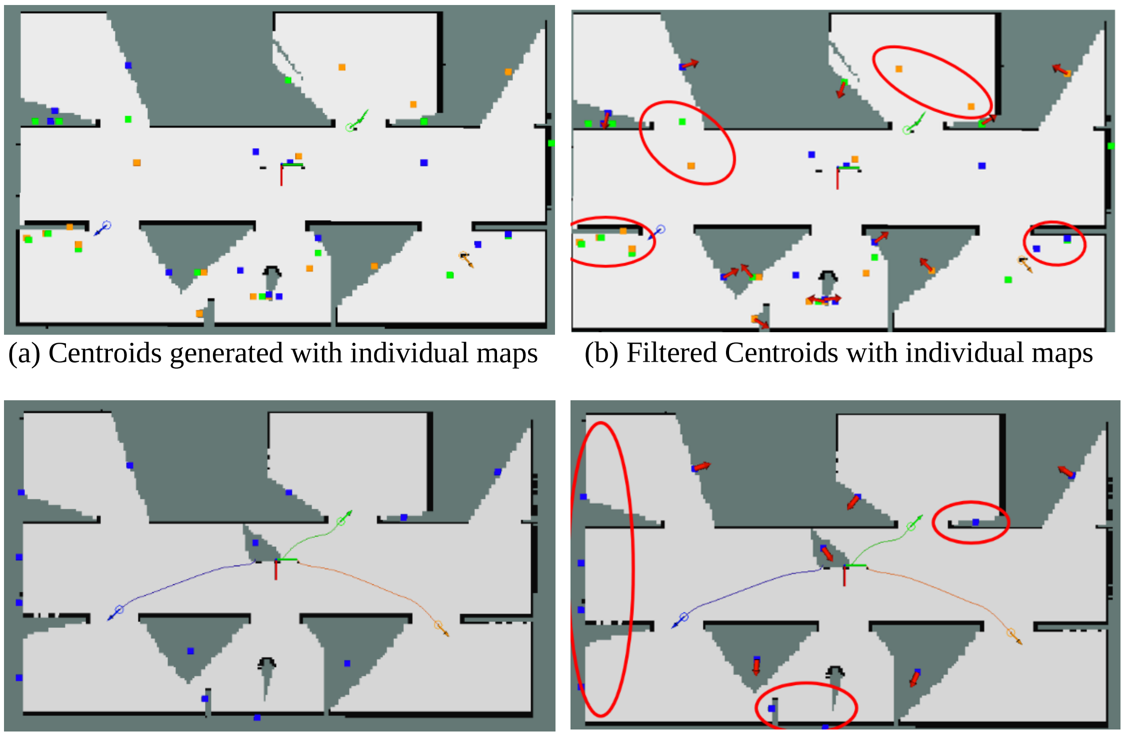
Figure 6: Algorithm 2

4.2. The Filter: Frontier Filtering and Compiling

With Section 4.1, every robot returns a list of centroid candidates, which could be based on their individual maps or merged map (see Figure 7(a) and (c) respectively). Due to inaccuracies in the *multirobot_map_merge* node, the use of individual maps is preferred. For coordination in the absence of a shared map, there is a need for frontier filtering to evaluate the viability of centroids and centroid compiling to update the explored areas. While the compilation of centroids is more useful for cases where individual maps are used, the filtering of frontiers has proven useful in eliminating unfeasible points even in merged map cases (see Figure 7(d)).

The filter maintains individual maps and works by searching in both positive and negative directions of the centroid's gradient for an obstacle in all maps. If no obstacle exists, then the number of unknown octile neighbours will determine the identity of the cell. If the number is below a threshold, the centroid is already explored by other robots and should be excluded. During the gradient search, the coordinates of the centroids are updated to the location that gives the highest gradient. Thus, this increases the accuracy of using octile neighbour in determining cells' identity.

Figure 7 illustrates the impact of the filter node on the final list of centroids for both the individual map (a)-(b) and merged map case (c)-(d). The square markers represent centroids generated by the individual detectors. The final feasible centroids are represented by red arrows that point in the direction of frontier. The circled regions for (b) and (d) represent centroids that were filtered.


(a) Centroids generated with individual maps (b) Filtered Centroids with individual maps

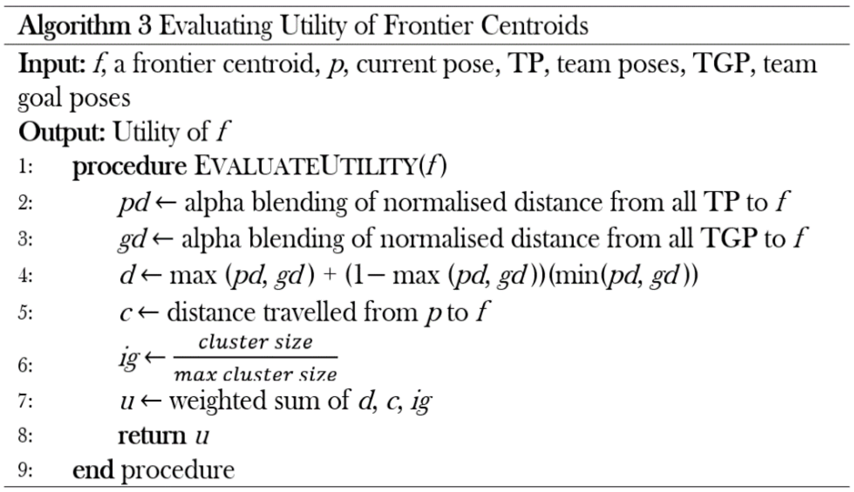
(c) Centroids generated with merged map (d) Filtered Centroids with merged map

Figure 7: Effect of Filter Node

Then, the centroids are compiled or clustered using a similar algorithm as in Section 4.1, with criteria to measure similarity as a function of both distance and gradients. Accounting for gradient becomes important to distinguish centroids in close proximities with each other (see Appendix D).

4.3. The Allocator: Coordinated Frontier Allocation

The coordination of the team hinges on the utility calculation for frontier allocation. The computation of utility is outlined in Algorithm 3, where utility is a weighted sum of discount (d), cost (c), and information gain (ig). The symbols f, p TP and TGP carry the same meaning as defined in Algorithm 3.



```
Algorithm 3 Evaluating Utility of Frontier Centroids
Input: f, a frontier centroid, p, current pose, TP, team poses, TGP, team goal poses
Output: Utility of f
1:  procedure EVALUATEUTILITY(f)
2:      pd ← alpha blending of normalised distance from all TP to f
3:      gd ← alpha blending of normalised distance from all TGP to f
4:      d ← max (pd, gd) + (1− max (pd, gd))(min(pd, gd))
5:      c ← distance travelled from p to f
6:      ig ← $\frac{cluster\ size}{max\ cluster\ size}$
7:      u ← weighted sum of d, c, ig
8:      return u
9:  end procedure
```

Figure 8: Algorithm 3

Discount ($\boldsymbol{d}$): represents the diminished visibility and utility of centroids due to the presence of other robots' sensors. While visibility is only diminished for centroids near selected targets in [1], the concept is extended to diminishing visibility of centroids near the robots. They are represented by the goal discount (gd_f) and pose discount (pd_f) respectively. Diminished visibility is the ratio of the distance between the centroid and the pose (TGP or TP) and laser range. gd_f and pd_f are computed as accumulation of diminished visibilities from all TGP and TP via alpha blending. For a robot team of size t, $gd_f = gd_t$ and $gd_1 = \left(1 - \frac{norm(TP_1, f)}{laser\ range}\right)$. pd_f is computed similarly. d_f is a function of gd_f and pd_f as in Algorithm 3 line 4.

$$gd_i = gd_{i-1} + (1 - gd_{i-1})\left(1 - \frac{norm(TGP_i, f)}{laser\ range}\right) \quad (1)$$

Cost ($\boldsymbol{c}$): represents the expected distance travelled from the current position to the centroid. A Dijkstra [23] planner from the navigation function (<u>*navfn*</u>) class is used to plan a path from p to f. For a path of size n containing map index, $x_i \dots x_n$, the cost depends on the occupancy cell value $M(x_i)$:

$$c = \sum_{i=1}^{n} c_i \begin{cases} 1 & if\ M(x_i) = 0 \\ m & otherwise \end{cases} \tag{2}$$

where m represents the additional cost in manoeuvring past an unknown or occupied cell. c is normalised by dividing it with the diagonal length of the environment to give c_f. Figure 9 shows the difference in path and accuracy between the previously proposed bresenham algorithm (red) and *navfn* (green).

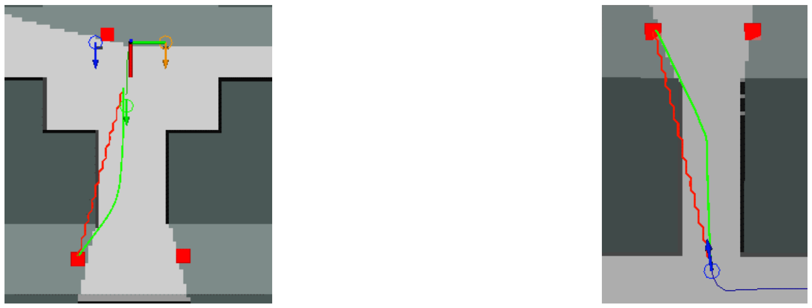


Figure 9: Paths Generated For Cost Calculation

Information gain ($\boldsymbol{ig}$): represents the expected area that can be uncovered by the robot at that centroid [4]. It is estimated by the size of the unknown area around the centroid or, in this case, by the cluster size for reduced computation. ig_f is normalised by taking the ratio of its cluster size to the maximum cluster size.

$$ig_f = \frac{size(f)}{\max\limits_{f_i \in F}\left(size(f_i)\right)} \tag{3}$$

Utility ($\boldsymbol{u}$): represents the weighted value of reaching the cell, where cell with the highest utility will be visited. Utility for a frontier cell u_f is defined as:

$$u_f = -(w_d \times d_f) - (w_c \times c_f) + (w_{ig} \times ig_f) \tag{4}$$

where w_d, w_c, w_{ig} represents the weights given to d_f, c_f, and ig_f, respectively. Appendix E shows the cost, discount, information gain, and utility for the centroids during a simulation.

4.4. Greedy Map Merging

The *multirobot_map_merge* generates errors when running (see Figure 11(a)). This is likely due to errors with frame transformations. Map merging can be alternatively achieved by transforming all maps to the global frame with the pre-defined transformation matrix and then taking the maximum value of the cell intersections. The transformation matrix, ${}^{F}T_{M}$, transforms coordinates in map frame, ${}^{M}P$, into the fixed (or global) frame ${}^{F}P$ (equation 5). ${}^{F}T_{M}$ accounts for rotation, as represented by $R(\theta)$, and translation, as represented by vector P (equation 6). θ refers to the M's yaw from F, while P is the translation of the M's origin from F's origin. Both θ and P are expressed in terms of the global frame.

$${}^{F}P = {}^{F}T_{M}\ {}^{M}P \tag{5}$$

$${}^{F}T_{M} = \begin{bmatrix} R(\theta) & P \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & p_1 \\ \sin\theta & \cos\theta & 0 & p_2 \\ 0 & 0 & 1 & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

The process of the map merging can be found in Figure 10.

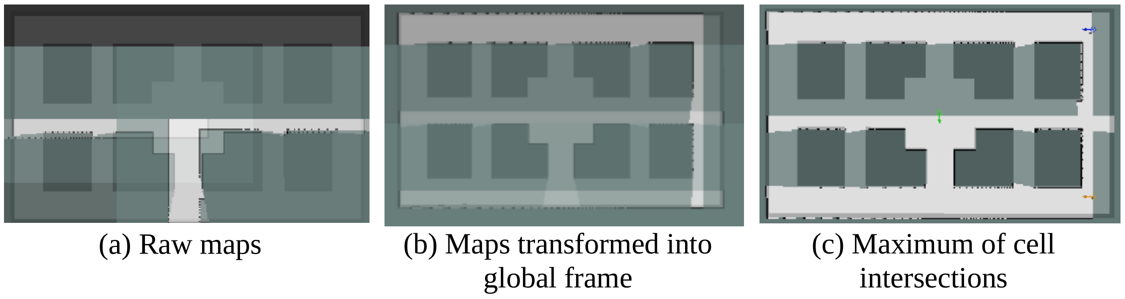

(a) Raw maps (b) Maps transformed into global frame (c) Maximum of cell intersections

Figure 10: Proposed Map Merging

The method has worked well for this 2D application with known initial positions and was used for the detector and for performance evaluation (see Figure 11).

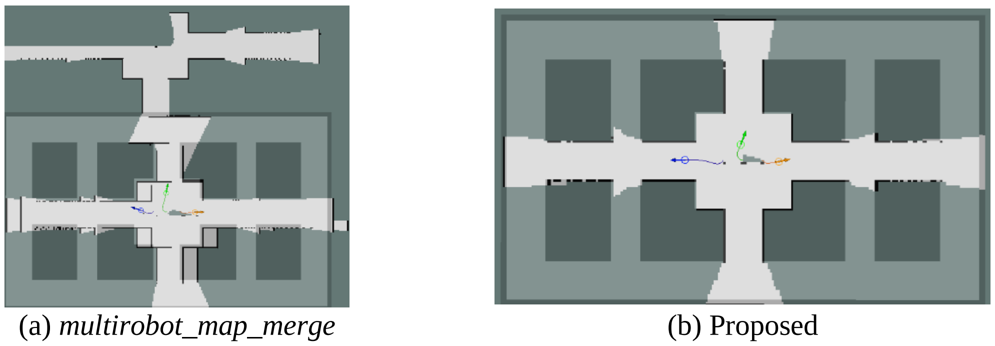

(a) *multirobot_map_merge* (b) Proposed

Figure 11: Maps Produced by *multirobot_map_merge* and Proposed Method

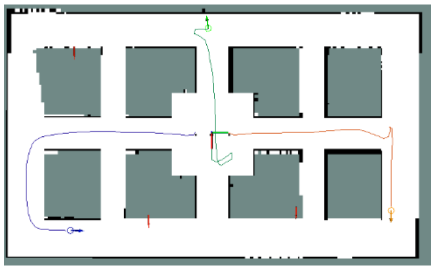
The ROS parameters used, along with significance and value, for the exploration module are detailed in Appendix F.

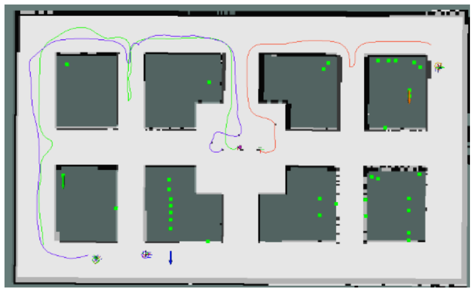
5. RESULTS

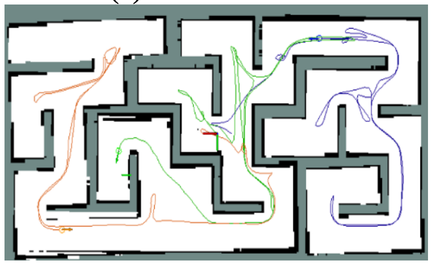
Each algorithm was tested 10 times in each environment, with a standardised team size of 3 (total simulations = 80). Prepared shell scripts were executed in the following order for the simulation: *env.sh* for the multi-robot simulator, *run.sh* for the exploration module, and *save.sh* for metrics measurement. The detailed procedure is found in Appendix G. Appendix H, I and J provides the documentation and links for the developed ROS packages titled *mrs_env simulator*, *cfe_module*, and *env_stats*.

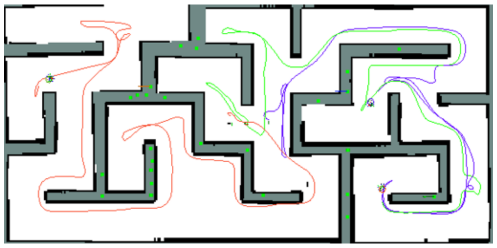
5.1. Trajectory of Robots in Simulated Environments

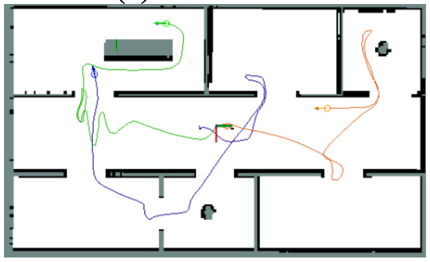
Figure 12 shows the final merged map and path taken for CFE (a-d) and RRT (e-h) for a sample run. The trajectories for all the simulations are presented in Appendix K.

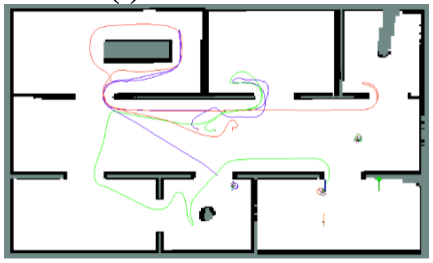

(a) CFE in Blocks

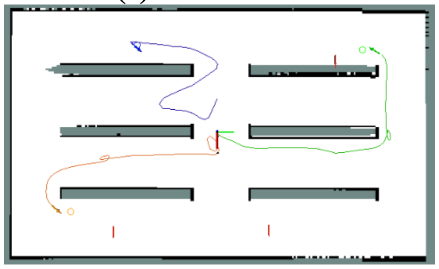

(e) RRT in Blocks


(b) CFE in Maze


(f) RRT in Blocks


(c) CFE in Office


(g) RRT in Office


(d) CFE in Warehouse

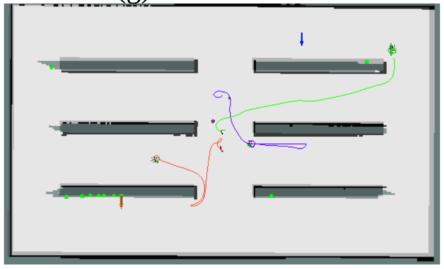
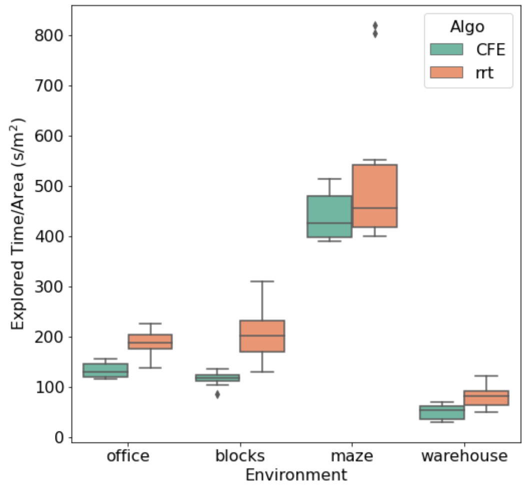

(h) RRT in Warehouse

Figure 12: Sample Trajectories for the Environments

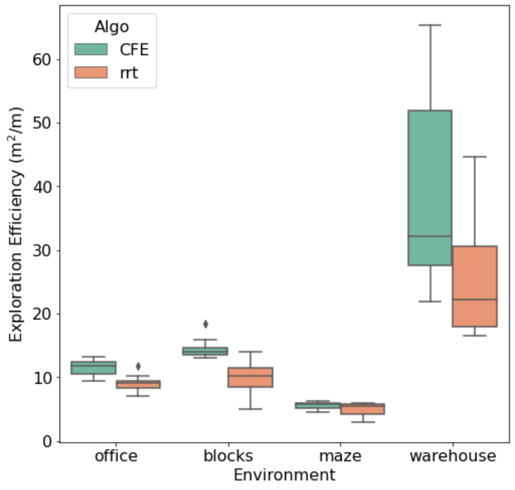
From initial inspection, coordination is attained in CFE and RRT as the robots were fairly distributed in the environment. While the trajectories of CFE diverge from the start, the trajectories of RRT tend to stay close before some divergence is observed. The effect of such trajectory on the performance of the exploration was measured and quantified with the performance metrics.

5.2. Evaluation of Performance against RRT-exploration

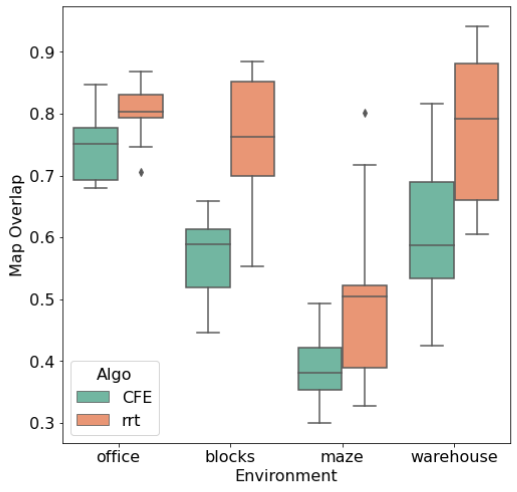
Exploration time per area covered, exploration efficiency, and map overlap were the performance metrics chosen for evaluation. Exploration time per area covered, or time/area and exploration efficiency were more apt due to the incomplete runs for certain environments (see Table XX). The performance of CFE and RRT for the three metrics is contrasted in Figure 13.



(a) Explored Time/Area



(b) Exploration Efficiency



(c) Map Overlap

Figure 13: Performance of CFE and RRT

CFE generally performs better than RRT, with lower exploration time/area, map overlap, and higher efficiency as seen in Table 1. Using T-Test, the observed difference has also been proven to be statistically significant with P-values smaller than 5% (i.e., reject that null hypothesis for data sets to be similar). An exception is the Maze environment where the p-value was found to be greater than 5% for time/area and efficiency metrics.

Table 1: Difference in performance between CFE and RRT

	% Diff				P-Value			
	B[1]	O	M	W	B	O	M	W
Time/Area (s/m^2)	-28.0	-43.9	-15.8	-37.6	6e-5	3e-4	0.070	0.001
Efficiency (m^2/m)	27.2	48.8	13.9	55.0	6e-4	4e-4	0.139	0.028
Map Overlap	-7.10	-25.0	-24.0	-22.2	0.014	1e-4	0.018	0.004

[1]: B: Block, O:Office, M:Maze, W:Warehouse

Out of 40 runs, RRT could not complete exploration for 5 runs, while CFE was able to complete all exploration (see Table 2).

Table 2: Incomplete Explorations

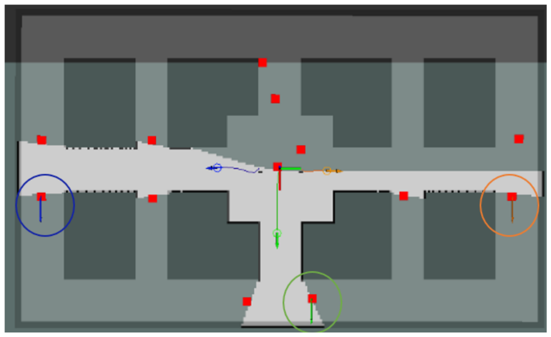
	Number of Incomplete Runs			
	B	O	M	W
CFE	0	0	0	0
RRT	3	0	2	0

6. DISCUSSION

This section explores the possible reasons for the differences in the performance of CFE and RRT in the environments. For simplicity, exploration time is treated to be analogous to exploration time/area for future sections.

6.1. Distribution of Robots

As observed in Section 5.1, robots in CFE are distributed as trajectories diverge from the start to uncover more unexplored regions, which subsequently guides better frontier selections. For example, if robots were to travel along the same corridor for the environment, Blocks, they may miss out exploring the other three corridors that branch into hallways, which can reduce the speed. The discount on centroids close to other robot's pose or goal poses used in the utility calculation repels robots to explore distinct regions and to only converge if the centroid has large information gain or low cost. Figure 14 shows some generated target points during CFE. While RRT also has a discount factor, it is a fixed value regardless of the closeness of the frontier points, which may not repel robots adequately (see Figure 14(a)). The frontier detection in RRT is also much slower as the tree is designed to grow over time: it detects a single frontier in each iteration (see Figure 14 (b)). The result is CFE is able to allocate robots to distinct frontier regions while RRT initially allocates robots to the same region due to the limited number of frontier points. Thus, CFE would benefit from faster exploration, higher efficiency, and less extent of map overlap.


(a) Blocks

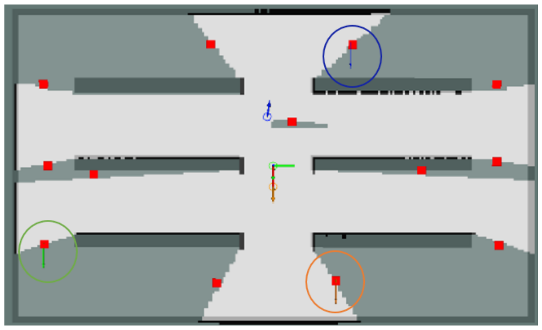
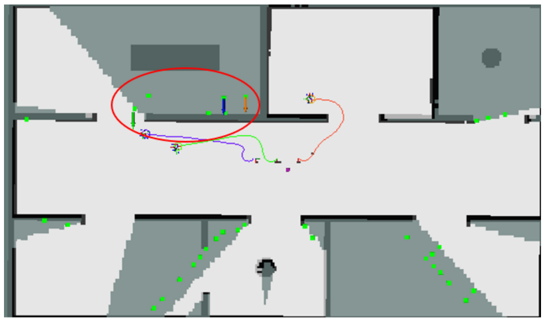

(b) Warehouse

Figure 14: Distributed allocation of robots for CFE


(a) Office

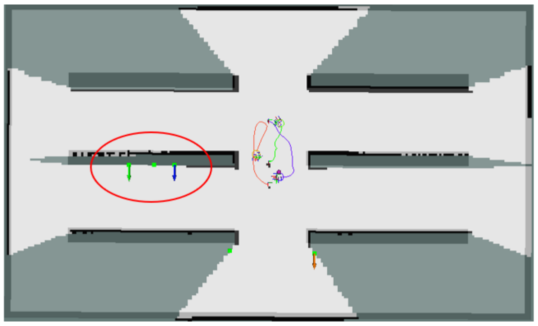
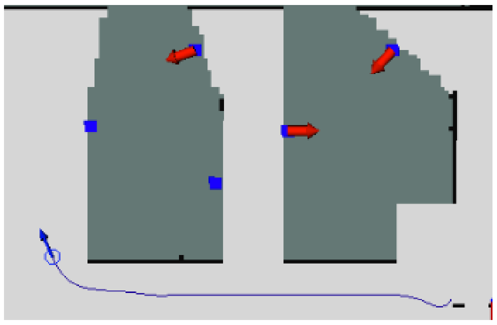

(b) Warehouse

Figure 15: Clustered allocation of robots for RRT

6.2. Detection of valid and feasible frontiers

The RRT can perform less ideally in situations where there are large obstacles (like Blocks and Office) as it increases the detection of infeasible frontiers. A challenge with frontier detection with updating sensor information is differentiating between frontiers, obstacle boundaries, and unknown space. Unknown spaces, such as space beyond the enclosed environment or space inside the perimeter of big obstacles, are frequently falsely detected as a frontier by RRT as it only checks if a point is unknown. These spaces are often inaccessible and do not contribute to exploration. On the other hand, through the detector and filter node, CFE generates valid centroid points that lie on the boundaries between known and unknown space. By computing points along the edge, there is less chance for unknown, inaccessible space to be detected as a frontier. Figure 16 shows the centroids generated by CFE (a) and RRT (b) for the same region. The circled region shows robots travelling to an inaccessible point. This likely increases exploration time and reduces efficiency as the robot will circle around the region. Without proper recovery behaviours, the robots are stuck and cannot complete the exploration, which results in incomplete runs for RRT.


(a) Feasible frontiers in CFE

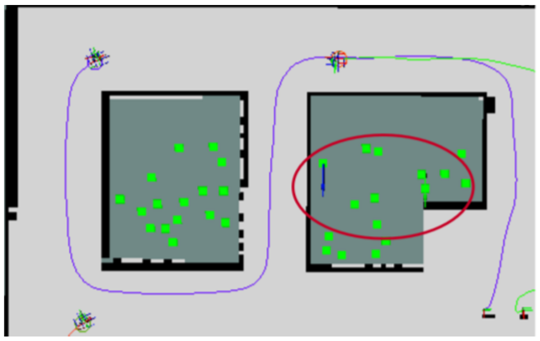

(b) Unfeasible frontiers in RRT

Figure 16: Detection of Feasible Frontiers

6.3. Limitation in Complex Environments

While CFE has performed better in most environments, its performance rivals that of RRT in the Maze environment. This is attributed to the decoupled system between the navigation and the merged map. There are two implications, the first is the underestimation of navigation cost, and the second is ineffective planned path due to lack of information on the environment. Currently, the ROS packages for navigation only consider the robot’s individual map and is unable to proceed efficiently to far frontier points detected by other robots as the planner lack the necessary information. While this applies to both CFE and RRT, CFE is more affected due to its nature to repel robots and identify distinct regions with little overlap in the map information. In a complex environment like the Maze, when the robot is allocated a point beyond its mapping information, the robot travels redundantly, increasing exploration cost and time. On the other hand, RRT bias frontiers with visibility and often allocate robots to closer frontiers, a strategic navigation technique in a complex environment. The trade-off is reduced distribution in other environments.

7. Conclusion

Through this project, a multi-robot test-simulator with three components: simulator, an exploration module, and performance evaluator, has been designed and tested as three separate ROS packages. Through the multi-robot simulator, a variable number of robots with communicative abilities can be spawned in desired environments. The exploration module, CFE, a variant of CME, has worked well in 2-D indoor environments when compared to another existing exploration module. As such, CFE serves as a proper benchmark for testing variants of CME with coordination strategies to fulfil other objectives. Lastly, the performance evaluator can track metrics of interest and describe the quality of exploration quantitatively.

Future works for each of the three components will be proposed. Firstly, a navigation stack that supports planning in merged map can be developed for better cost estimation and path planning. This increases the support for multi-robot systems by utilising its increased information gain. Secondly, other frontier detection methods like the Fast Frontier Detector in [26] can be tested for fast detection, which would be crucial for deployment in large environments. Lastly, image processing techniques should be utilised to transform and match the occupancy grids. Map merging techniques, as in [24], can increase the robustness of existing map analysis during performance evaluation and reduce the need for known initial positions.

LIST OF REFERENCES

[1] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on robotics,* vol. 21, no. 3, pp. 376-386, 2005.

[2] A. Rao, F. B. Abdesslem, A. Lindgren, and A. Ziviani, "Team communication strategy for collaborative exploration by autonomous vehicles," in *2016 IEEE International Conference on Communications (ICC)*, 2016: IEEE, pp. 1-6.

[3] R. Simmons *et al.*, "Coordination for multi-robot exploration and mapping," in *Aaai/Iaai*, 2000, pp. 852-858.

[4] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart, "Distributed multirobot exploration and mapping," *Proceedings of the IEEE,* vol. 94, no. 7, pp. 1325-1339, 2006.

[5] A. Marjovi, J. G. Nunes, L. Marques, and A. De Almeida, "Multi-robot exploration and fire searching," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009: IEEE, pp. 1929-1934.

[6] Y. Han, D. Li, J. Chen, X. Yang, Y. Hu, and G. Zhang, "A multi-robots task allocation algorithm based on relevance and ability with group collaboration," *International Journal of Intelligent Engineering and Systems,* vol. 3, no. 2, pp. 33-41, 2010.

[7] L. E. Parker, "Current state of the art in distributed autonomous mobile robotics," *Distributed Autonomous Robotic Systems 4,* pp. 3-12, 2000.

[8] Z. Yan, L. Fabresse, J. Laval, and N. Bouraqadi, "Building a ros-based testbed for realistic multi-robot simulation: Taking the exploration as an example," *Robotics,* vol. 6, no. 3, p. 21, 2017.

[9] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Proceedings of the second international conference on Autonomous agents*, 1998, pp. 47-53.

[10] R. Cipolleschi, M. Giusto, A. Q. Li, and F. Amigoni, "Semantically-informed coordinated multirobot exploration of relevant areas in search and rescue settings," in *2013 European Conference on Mobile Robots*, 2013: IEEE, pp. 216-221.

[11] S. Oßwald, M. Bennewitz, W. Burgard, and C. Stachniss, "Speeding-up robot exploration by exploiting background information," *IEEE Robotics and Automation Letters,* vol. 1, no. 2, pp. 716-723, 2016.

[12] 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)*, 2017: IEEE, pp. 1396-1402.

[13] J. Vazquez and C. Malcolm, "Distributed multirobot exploration maintaining a mobile network," in *2004 2nd International IEEE Conference on'Intelligent Systems'. Proceedings (IEEE Cat. No. 04EX791)*, 2004, vol. 3: IEEE, pp. 113-118.

[14] M. N. Rooker and A. Birk, "Multi-robot exploration under the constraints of wireless networking," *Control Engineering Practice,* vol. 15, no. 4, pp. 435-445, 2007.

[15] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, 2009, vol. 3, no. 3.2: Kobe, Japan, p. 5.

[16] G. Echeverria *et al.*, "Simulating complex robotic scenarios with MORSE," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2012: Springer, pp. 197-208.

[17] Z. Yan, L. Fabresse, J. Laval, and N. Bouraqadi, "Team size optimization for multi-robot exploration," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2014: Springer, pp. 438-449.

[18] C. Nieto-Granda, J. G. Rogers III, and H. I. Christensen, "Coordination strategies for multi-robot exploration and mapping," *The International Journal of Robotics Research,* vol. 33, no. 4, pp. 519-533, 2014.

[19] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A Review of Physics Simulators for Robotic Applications," *IEEE Access*, 2021.

[20] S. Thrun, "Simultaneous localization and mapping," in *Robotics and cognitive approaches to spatial mapping*: Springer, 2007, pp. 13-41.

[21] X. Zhang, J. Lai, D. Xu, H. Li, and M. Fu, "2D LiDAR-based SLAM and path planning for indoor rescue using mobile robots," *Journal of Advanced Transportation,* vol. 2020, 2020.

[22] F. Gul, W. Rahiman, and S. S. Nazli Alhady, "A comprehensive study for robot navigation techniques," *Cogent Engineering*, vol. 6, no. 1, p. 1632046, 2019.

[23] J. C. Latombe, *Robot Motion Planning*. Springer US, 1991.

[24] S. Yu, C. Fu, A. K. Gostar, and M. Hu, "A Review on Map-Merging Methods for Typical Map Types in Multiple-Ground-Robot SLAM Solutions," *Sensors*, vol. 20, no. 23, p. 6988, 2020.

[25] J. Hörner, "Map-merging for multi-robot system," 2016.

[26] 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.

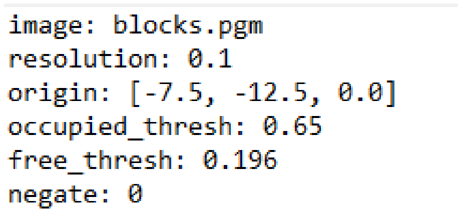
APPENDICES

Appendix A: Creation of Gazebo Environments

Pre-requisites: map_server, map2gazebo (https://github.com/shilohc/map2gazebo)

Step 1: Create outline of environment in Portable Gray Format (.pgm). GNU Image Manipulation Program (GIMP) was used.

Step 2: Create a yaml file describing characteristics of the environment.



```
image: blocks.pgm
resolution: 0.1
origin: [-7.5, -12.5, 0.0]
occupied_thresh: 0.65
free_thresh: 0.196
negate: 0
```

Example: blocks.yaml file for blocks.pgm (created in step 1)

Step 3: In the directory containing the yaml file, run:

```
rosrun map_server map_server [filename]
```

e.g. [filename] = blocks.yaml

This converts the image file into an occupancy grid that is published as /map.

Step 4: In the catkin workspace containing map2gazebo, run:

```
roslaunch map2gazebo map2gazebo.launch
```

The resulting gazebo environment overwrites the map.stl found in map2gazebo/models/map/meshes/.

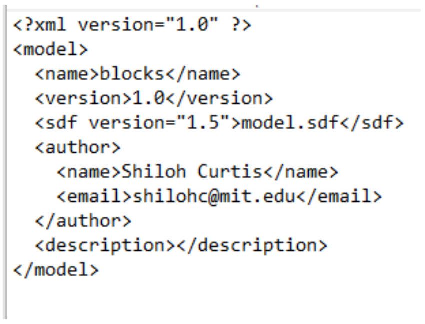
Step 5: Rename the map.stl accordingly, changing the corresponding tags in model.config and model.sdf.

The “map2gazebo” package detects features, specifically contours, using the OpenCV Library. The level of hierarchy can also be to be adjusted to detect all contours (i.e., the code uses cv.RETR_CCOMP which classifies contours into two-level hierarchy.

However, this ignores the inner obstacles in office rooms and maze walls. To circumvent that, cv.RETR_TREE could be used instead to can identify all contours.

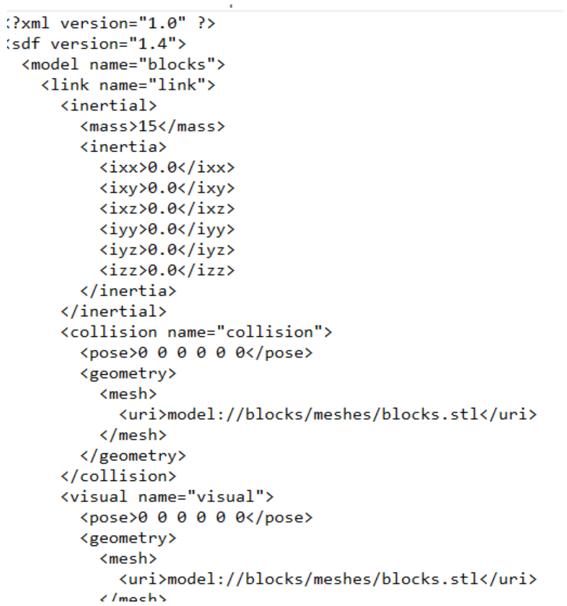
```
gazebo
  -  models
       -  blocks
            -  meshes/block.stl
            -  model.config
            -  model.sdf
       -  maze
       -  office
       -  warehouse
  -  urdf
  -  worlds
       -  blocks.sdf
       -  maze.sdf
       -  office.sdf
       -  warehouse.sdf
```

Figure A1: Structure in “mrs_env_simulator” for gazebo environment simulations:



```
<?xml version="1.0" ?>
<model>
  <name>blocks</name>
  <version>1.0</version>
  <sdf version="1.5">model.sdf</sdf>
  <author>
    <name>Shiloh Curtis</name>
    <email>shilohc@mit.edu</email>
  </author>
  <description></description>
</model>
```

(a) model.config

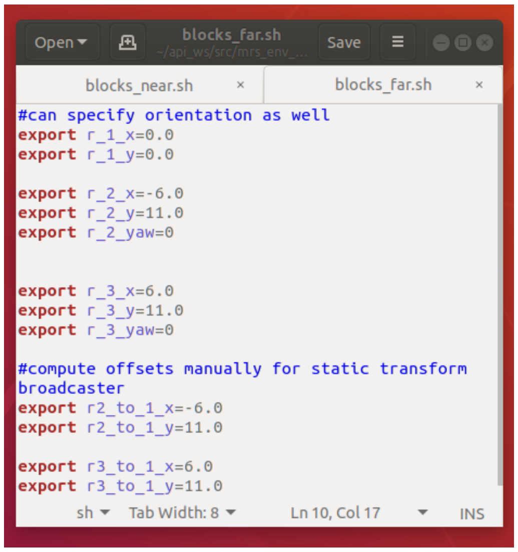


```
:?xml version="1.0" ?>
:sdf version="1.4">
  <model name="blocks">
    <link name="link">
      <inertial>
        <mass>15</mass>
        <inertia>
          <ixx>0.0</ixx>
          <ixy>0.0</ixy>
          <ixz>0.0</ixz>
          <iyy>0.0</iyy>
          <iyz>0.0</iyz>
          <izz>0.0</izz>
        </inertia>
      </inertial>
      <collision name="collision">
        <pose>0 0 0 0 0 0</pose>
        <geometry>
          <mesh>
            <uri>model://blocks/meshes/blocks.stl</uri>
          </mesh>
        </geometry>
      </collision>
      <visual name="visual">
        <pose>0 0 0 0 0 0</pose>
        <geometry>
          <mesh>
            <uri>model://blocks/meshes/blocks.stl</uri>
```

(b) model.sdf

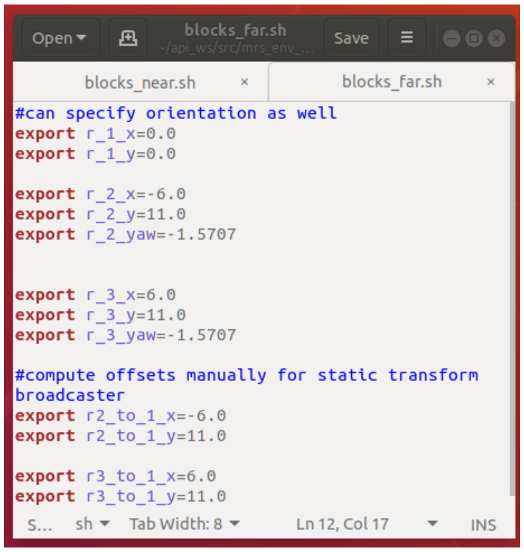
Figure A2: Sample files found in /models/blocks/

Appendix B: Pre-set Environment Configurations



```
#can specify orientation as well
export r_1_x=0.0
export r_1_y=0.0

export r_2_x=-6.0
export r_2_y=11.0
export r_2_yaw=0


export r_3_x=6.0
export r_3_y=11.0
export r_3_yaw=0

#compute offsets manually for static transform
broadcaster
export r2_to_1_x=-6.0
export r2_to_1_y=11.0

export r3_to_1_x=6.0
export r3_to_1_y=11.0
```

(a) near



```
#can specify orientation as well
export r_1_x=0.0
export r_1_y=0.0

export r_2_x=-6.0
export r_2_y=11.0
export r_2_yaw=-1.5707


export r_3_x=6.0
export r_3_y=11.0
export r_3_yaw=-1.5707

#compute offsets manually for static transform
broadcaster
export r2_to_1_x=-6.0
export r2_to_1_y=11.0

export r3_to_1_x=6.0
export r3_to_1_y=11.0
```

(b) far

Figure B1: Shell script file containing initial configurations for Blocks

The resulting pose of the robot (represented by coloured arrows) simulated for all the environments are presented in Figure B2.

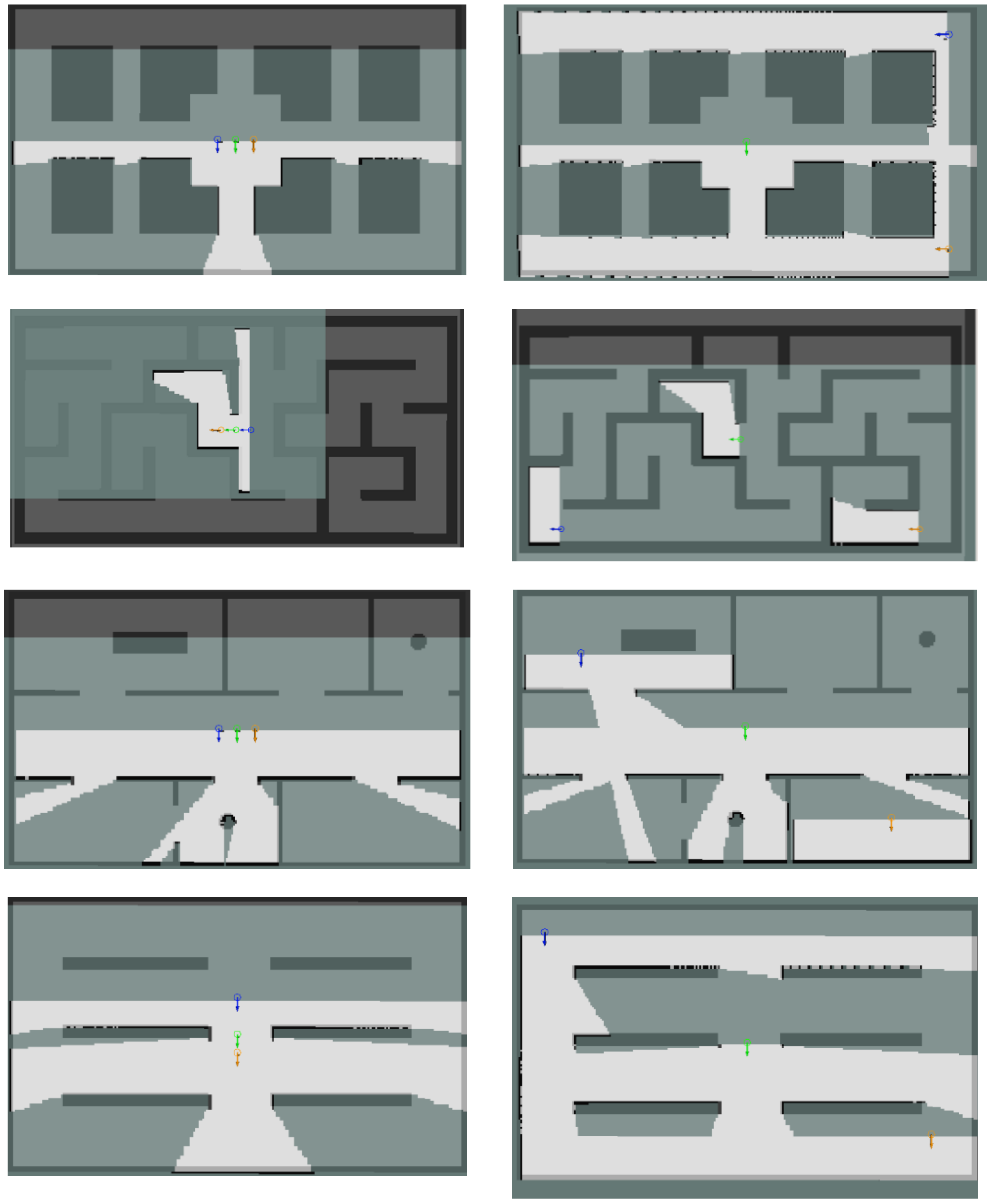
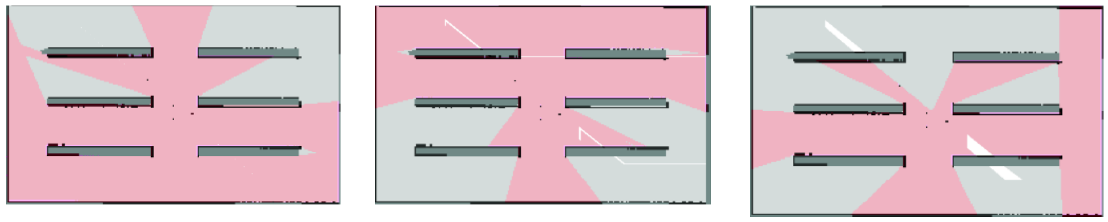

(a) near (b) far

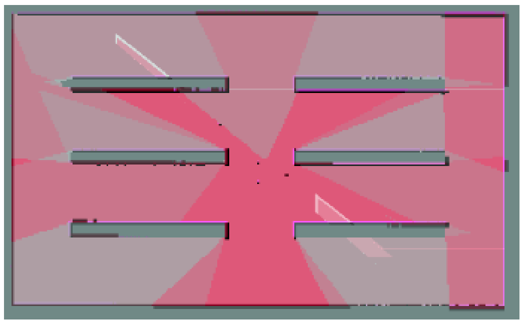
Figure B2: Initial Pose (near and far) of robots in Blocks, Maze, Office, and Warehouse environments

Appendix C: Calculation of Map Overlap

Map Overlap is calculated as the ratio of the overlapped area to total explorable area. To prevent double counting, the ratio of non-overlap area/total area is found instead, and Map Overlap is the complement of that ratio.

After superimposing the maps, if an area is known by two or more robots, it will be registered as unknown in the merged map. The final non-overlap area is the total known area in the merged map. The source code is in Figure C2.


(a) Red areas represent map built by each robot


(b) Superimposition of all maps

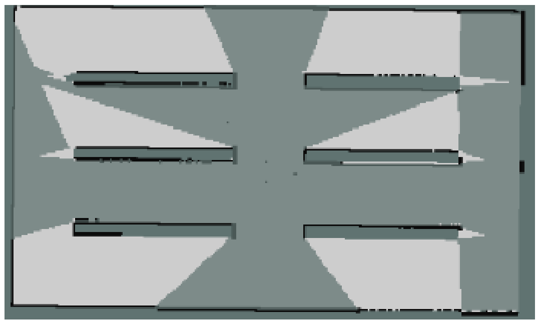
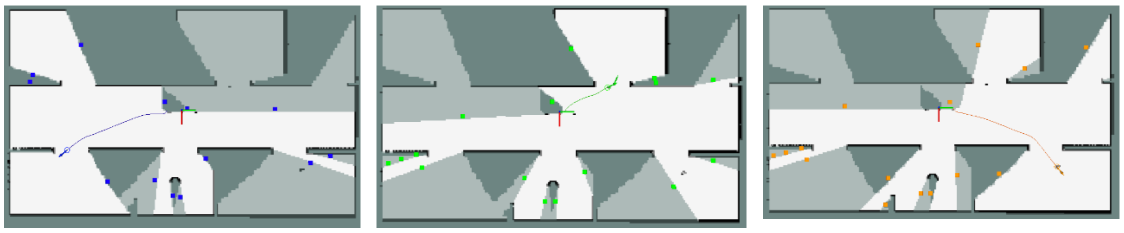

(c) White areas represent the total non-overlap area in the merged map

Figure C1: Visualisation of Map Overlap

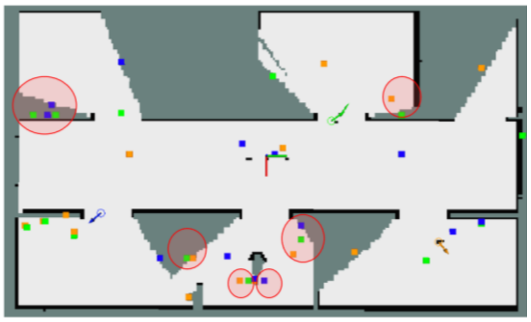
```
void calculateInternalOverlap() {

    geometry_msgs::Point globalOrigin = maps[3].info.origin.position;
    nav_msgs::OccupancyGrid combined = maps[3];

    //Extracting Map Information
    double res = combined.info.resolution;
    int width = combined.info.width;
    int height = combined.info.height;

    //Superimposing grids
    for (int mapit = 0; mapit < numberMaps-1; mapit++ ){
        geometry_msgs::Point base = maps[mapit].info.origin.position;

        for (int i = 0; i < width; i ++){
            for (int j = 0; j < .height; j++){
                int combinedidx = coordToIndex(i, j,width);
                if (combined.data[combinedidx] < 0) continue;

                int baseX = i + (globalOrigin.x - base.x)/res;
                int baseY = j + (globalOrigin.y - base.y)/res;
                int baseIdx;
                int cellValue = -1;
                int baseW = maps[mapit].info.width;
                int baseH = maps[mapit].info.height;
                if (inMap(baseX, baseY, baseW, baseH)){
                    baseIdx = coordToIndex(baseX, baseY, baseW);
                    cellValue = maps[mapit].data[baseIdx];
                } else {continue;}

                if (cellValue < 0) continue;

                for (int t = mapit+1; t < numberMaps-1; t++){
                    geometry_msgs::Point local =
maps[t].info.origin.position;
                    int ownX = i + (globalOrigin.x - local.x)/res;
                    int ownY = j + (globalOrigin.y - local.y)/res;
                    int ownW =  maps[t].info.width;
                    int ownH =  maps[t].info.height;

                    if (inMap(ownX, ownY,ownW,ownH)){
                        int ownIdx = coordToIndex(ownX, ownY, ownW);
                        if (maps[t].data[ownIdx] == cellValue){
                            combined.data[combinedidx] = -100;
                            maps[t].data[ownIdx] = -100;
                            maps[mapit].data[baseIdx] = -100;
                        }
                    }

                }
            }
        }
        maps[3] = combined;

    }

    //Calculating Overlaps
    for (int mapit = 0; mapit < numberMaps; mapit ++){
        int totalCells = 0;
        nav_msgs::OccupancyGrid tgt = maps[mapit];
        int width = tgt.info.width;
        int height =  tgt.info.height;
        for (int i = 0; i < width; i ++){
            for (int j = 0; j < height; j++){
                int idx = coordToIndex(i,j,width);
                if (tgt.data[idx] == -1) continue;
                totalCells +=1;
                if (tgt.data[idx] == -100){
                    similarities[mapit] += 1;
                    tgt.data[idx] = -1;
                }
            }
        }
        //Record Map Overlap
        overlapmaps[mapit] = tgt;
        similarities[mapit] = similarities[mapit]/totalCells;

    }
}
```

Figure C2: Source Code for Calculation of Map Overlap

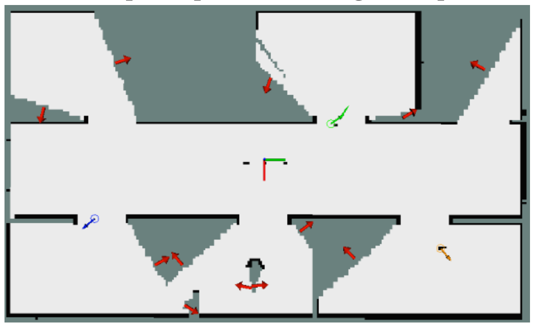
Appendix D: Clustering for Individual Map Use Case



(a) Centroids generated by individual robots superimposed on merged map



(b) All centroids generated. Circled regions represent possible clusters



(c) Red arrows point towards the gradient of the frontier and also marks the position of final (clustered and filtered) centroids

Figure D1: Effect of The Filter as mentioned in Section 4.2

Appendix E: Utility of Centroids

The centroids in Figure D1 have been ranked according to Utility Calculation rule in Section 4.3. The effect of pose discounts and goal discounts are evident as centroids 8, 10, 11, 12 all have higher discounts due to its proximity to the blue robot's pose and goal.

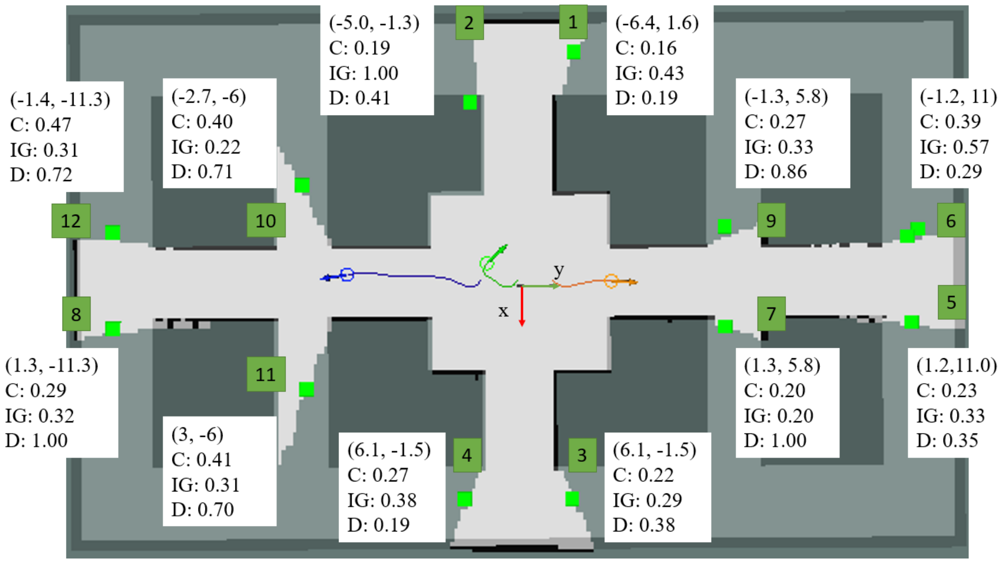


Figure E1: Sample Utility Calculation for Centroids

Appendix F: ROS Parameters, Definition and Default Values

The ROS parameters are defined in 2d_frontier_exploration.yaml in the *mrs_env_simulator* package.

Frontier Detection and Clustering:

- "frontier_thres": magnitude of gradient for a frontier (default: 4)
- "upper_thres": magnitude of gradient for cell to be an obstacle (default: 8)
- "cluster_rad": distance between frontier points to be clustered (default: 0.5)

Frontier Filtering and Compiling:

- "unknown_thres": total unknown neighbours for unknown cell (default: 2)
- "cluster_rad_m": distance between centroids to be clustered (default: 1)
- "cluster_grad": difference between gradients to be clustered (default: 0.5).
  The gradient is expressed as angle in radians.

Coordinated Frontier Allocation:

- "info_rad": analogous to $laser\ range$ (default: 10)
- "obs_mult": additional cost incurred for unknown or occupied cell (default: 5)
- "d_weight": weight for discount in utility (default: 1)
- "c_weight": weight for cost in utility (default: 1.5)
- "i_weight": weight for information gain in utility (default: 0.8)

Appendix G: Simulation Procedure

Pre-requisite: *mrs_env_simulator*, *cfe_module*, *env_stats*, *fyp_commands*

Pre-Simulation Set-up

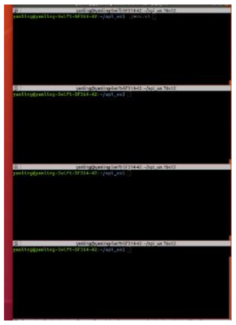
Create a catkin workspace and download the first three packages into the src folder. An additional folder titled *store* can be created for storage of map related files produced in step 6. The *.sh files can be obtained by pasting the files from *fyp_commands* into the workspace. See Figure G1 for recommended file structure.

```
catkin_ws
    -   devel
    -   build
    -   src
            -   cfe_module
            -   env_stats
            -   mrs_env_simulator

    -   store
    -   *.sh
```

Figure G1: Recommended file structure for workspace

Simulation Procedure

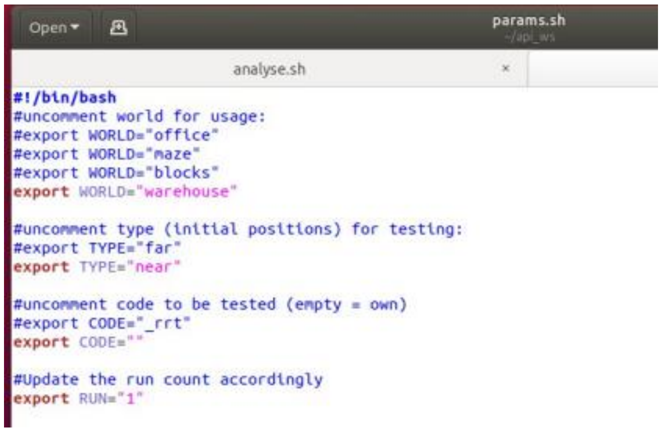
Step 1: In the catkin_ws, open up 4 terminals (terminator was used in this case). If not already done, run this command: chmod +x *.sh



```
yanling@yanling-Swift-SF314-42: ~/api_ws 78x12
yanling@yanling-Swift-SF314-42:~/api_ws$ ./env.sh

yanling@yanling-Swift-SF314-42: ~/api_ws 78x12
yanling@yanling-Swift-SF314-42:~/api_ws$

yanling@yanling-Swift-SF314-42: ~/api_ws 78x12
yanling@yanling-Swift-SF314-42:~/api_ws$

yanling@yanling-Swift-SF314-42: ~/api_ws 78x12
yanling@yanling-Swift-SF314-42:~/api_ws$
```

Figure G2: 4 empty terminals created with *terminator* in the catkin_ws

Step 2: Bring up the params.sh file which contains details about the simulation.



```
#!/bin/bash
#uncomment world for usage:
#export WORLD="office"
#export WORLD="maze"
#export WORLD="blocks"
export WORLD="warehouse"

#uncomment type (initial positions) for testing:
#export TYPE="far"
export TYPE="near"

#uncomment code to be tested (empty = own)
#export CODE="_rrt"
export CODE=""

#Update the run count accordingly
export RUN="1"
```

Figure G3: params.sh

WORLD refers to one of the four simulation environments.

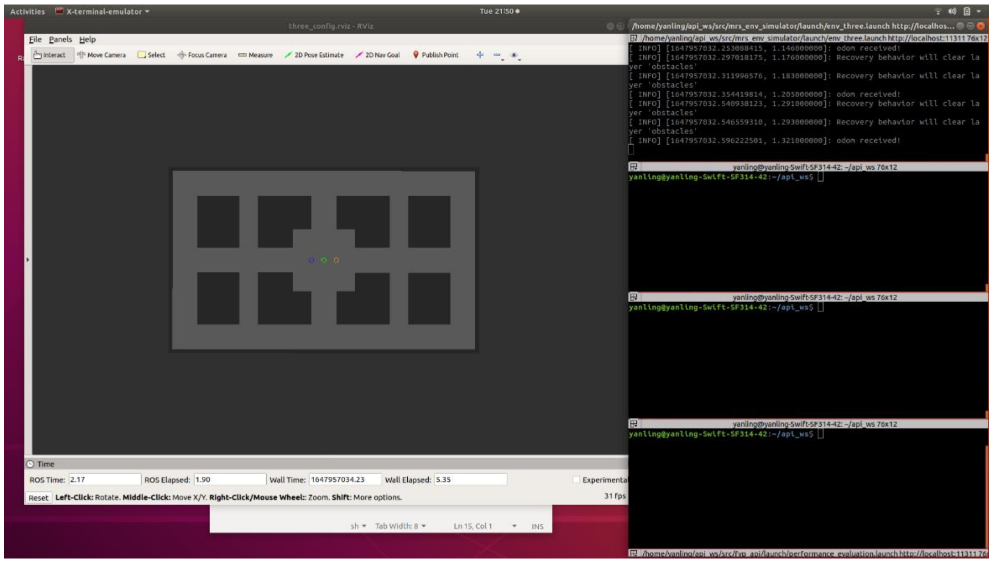
TYPE refers to the initial configuration.

CODE refers to the algorithm that is being tested.

RUN refers to the simulation count (e.g., 1st run, 5th run)

Uncomment the lines for usage.

Step 3: In one terminal, run the command: ./env.sh



```
/home/yanling/api_ws/src/mrs_env_simulator/launch/env_three.launch http://localhost:11311 76x12
[ INFO] [1647957032.253088415, 1.146000000]: odom received!
[ INFO] [1647957032.297018175, 1.176000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.311996576, 1.183000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.354419814, 1.205000000]: odom received!
[ INFO] [1647957032.540938123, 1.291000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.546559310, 1.293000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.596222501, 1.321000000]: odom received!
```

```
yanling@yanling-Swift-SF314-42: ~/api_ws 76x12
yanling@yanling-Swift-SF314-42:~/api_ws$
```

```
yanling@yanling-Swift-SF314-42: ~/api_ws 76x12
yanling@yanling-Swift-SF314-42:~/api_ws$
```

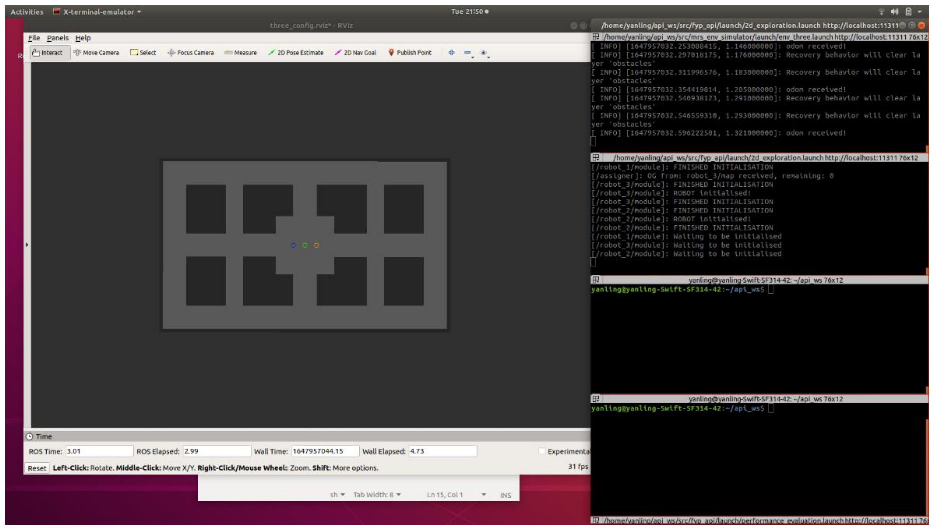
```
yanling@yanling-Swift-SF314-42: ~/api_ws 76x12
yanling@yanling-Swift-SF314-42:~/api_ws$
```

```
ROS Time: 2.17    ROS Elapsed: 1.90    Wall Time: 1647957034.23    Wall Elapsed: 5.35
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
```

Figure G4: Screenshot of interface after ./env.sh is run.

The environment will be brought up with ./env.sh. The Gazebo simulation has been suppressed in the env_three.launch file (in *mrs_env_simulator*) for speed and can be enabled by changing the <*gui*> tag.
Environment is fully simulated after "odom_received" appears as the last line.

Step 4: In another terminal, run the command: ./run.sh



```
/home/yanling/api_ws/src/mrs_env_simulator/launch/env_three.launch http://localhost:11311 76x12
[ INFO] [1647957032.253088415, 1.146000000]: odom received!
[ INFO] [1647957032.297018175, 1.176000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.311996576, 1.183000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.354419814, 1.285000000]: odom received!
[ INFO] [1647957032.540938123, 1.291000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.546559310, 1.293000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.596222501, 1.321000000]: odom received!
```

```
/home/yanling/api_ws/src/fyp_api/launch/2d_exploration.launch http://localhost:11311 76x12
[/robot_1/module]: FINISHED INITIALISATION
[/assigner]: OG from: robot_3/map received, remaining: 0
[/robot_3/module]: FINISHED INITIALISATION
[/robot_3/module]: ROBOT initialised!
[/robot_3/module]: FINISHED INITIALISATION
[/robot_2/module]: FINISHED INITIALISATION
[/robot_2/module]: ROBOT initialised!
[/robot_2/module]: FINISHED INITIALISATION
[/robot_1/module]: Waiting to be initialised
[/robot_3/module]: Waiting to be initialised
[/robot_2/module]: Waiting to be initialised
```

```
yanling@yanling-Swift-SF314-42:~/api_ws$
```

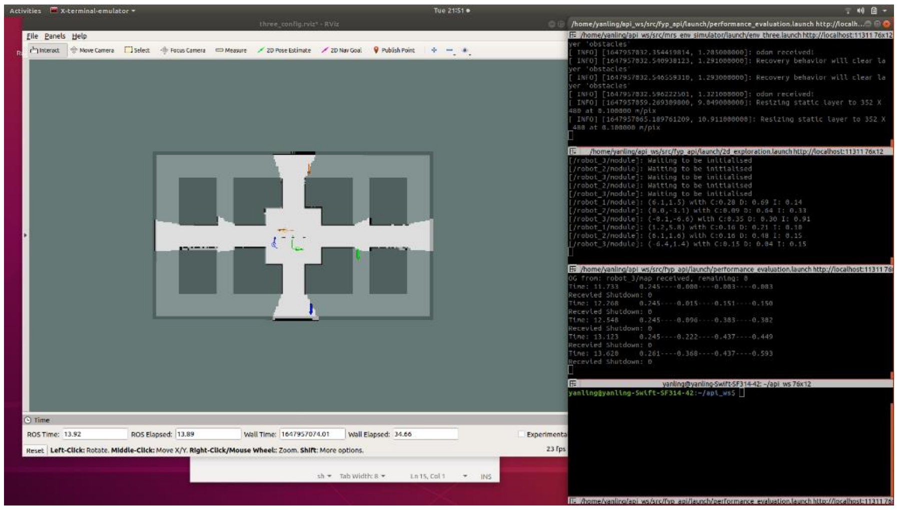
```
yanling@yanling-Swift-SF314-42:~/api_ws$
```

ROS Time: 3.01 ROS Elapsed: 2.99 Wall Time: 1647957044.15 Wall Elapsed: 4.73

Figure G5: Screenshot of interface after ./run.sh is run.

This starts the exploration with CFE by launching the necessary nodes in the *cfe_module*. The functionality of CFE is described in Section 4 and is characterised by the nodes which are: the *detector*, the *filter* and the *allocator*. The robots will start moving towards their target points through the *move_base_node*.

Step 5: In another terminal, run this command: ./stats.sh



```
/home/yanling/api_ws/src/mrs_env_simulator/launch/env_three.launch http://localhost:11311 76x12
yer 'obstacles'
[ INFO] [1647957832.354419814, 1.285000000]: odom received!
[ INFO] [1647957832.540938123, 1.291000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957832.546559310, 1.293000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957832.596222501, 1.321000000]: odom received!
[ INFO] [1647957859.269389800, 9.049000000]: Resizing static layer to 352 X
480 at 0.100000 m/pix
[ INFO] [1647957865.189761209, 10.911000000]: Resizing static layer to 352 X
 480 at 0.100000 m/pix
```

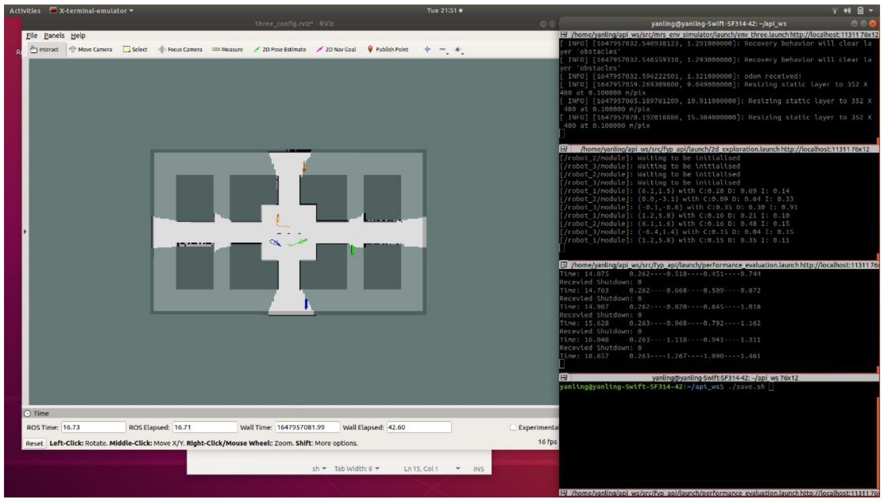
```
/home/yanling/api_ws/src/fyp_api/launch/2d_exploration.launch http://localhost:11311 76x12
[/robot_3/module]: Waiting to be initialised
[/robot_2/module]: Waiting to be initialised
[/robot_3/module]: Waiting to be initialised
[/robot_2/module]: Waiting to be initialised
[/robot_3/module]: Waiting to be initialised
[/robot_1/module]: (6.1,1.5) with C:0.28 D: 0.69 I: 0.14
[/robot_2/module]: (0.0,-3.1) with C:0.09 D: 0.64 I: 0.33
[/robot_3/module]: (-0.1,-6.6) with C:0.35 D: 0.30 I: 0.91
[/robot_1/module]: (1.2,5.8) with C:0.16 D: 0.21 I: 0.10
[/robot_2/module]: (6.1,1.6) with C:0.16 D: 0.48 I: 0.15
[/robot_3/module]: (-6.4,1.4) with C:0.15 D: 0.04 I: 0.15
```

```
/home/yanling/api_ws/src/fyp_api/launch/performance_evaluation.launch http://localhost:11311 76
OG from: robot_3/map received, remaining: 0
Time: 11.733     0.245----0.080----0.083----0.083
Receviled Shutdown: 0
Time: 12.268     0.245----0.015----0.151----0.150
Receviled Shutdown: 0
Time: 12.548     0.245----0.096----0.383----0.382
Receviled Shutdown: 0
Time: 13.123     0.245----0.222----0.437----0.449
Receviled Shutdown: 0
Time: 13.628     0.261----0.368----0.437----0.593
Receviled Shutdown: 0
```

```
yanling@yanling-Swift-SF314-42: ~/api_ws 76x12
yanling@yanling-Swift-SF314-42:~/api_ws$
```

Figure G6: Screenshot after ./stats.sh is run.

This kickstarts the collection of the different metrics (e.g., position, distance travelled, map completeness) by launching the *evaluator* node in *env_stats*. The merged map is also produced by the *evaluator* for visualisation of the overall exploration efforts. This merged map is superimposed over the ground truth.



```
/home/yanling/api_ws/src/mrs_env_simulator/launch/env_three.launch http://localhost:11311 76x12
[ INFO] [1647957032.540938123, 1.291000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.546559310, 1.293000000]: Recovery behavior will clear la
yer 'obstacles'
[ INFO] [1647957032.596222501, 1.321000000]: odom received!
[ INFO] [1647957059.269389800, 9.049000000]: Resizing static layer to 352 X
480 at 0.100000 m/pix
[ INFO] [1647957065.189761209, 10.911000000]: Resizing static layer to 352 X
 480 at 0.100000 m/pix
[ INFO] [1647957078.192016886, 15.384000000]: Resizing static layer to 352 X
 480 at 0.100000 m/pix
```

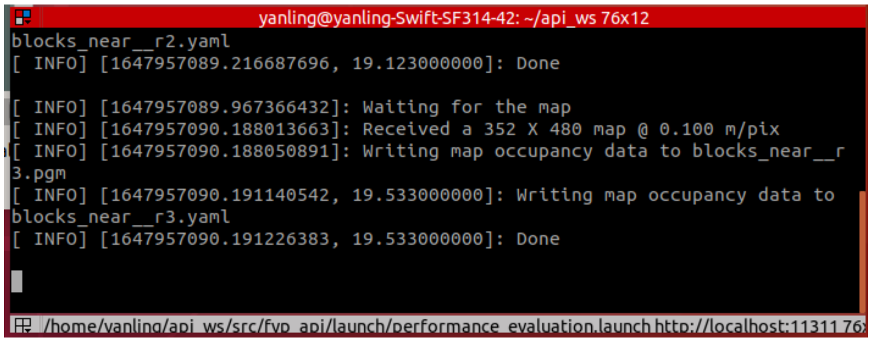
```
/home/yanling/api_ws/src/fyp_api/launch/2d_exploration.launch http://localhost:11311 76x12
[/robot_2/module]: Waiting to be initialised
[/robot_3/module]: Waiting to be initialised
[/robot_2/module]: Waiting to be initialised
[/robot_3/module]: Waiting to be initialised
[/robot_1/module]: (6.1,1.5) with C:0.28 D: 0.69 I: 0.14
[/robot_2/module]: (0.0,-3.1) with C:0.09 D: 0.64 I: 0.33
[/robot_3/module]: (-0.1,-6.6) with C:0.35 D: 0.30 I: 0.91
[/robot_1/module]: (1.2,5.8) with C:0.16 D: 0.21 I: 0.10
[/robot_2/module]: (6.1,1.6) with C:0.16 D: 0.48 I: 0.15
[/robot_3/module]: (-6.4,1.4) with C:0.15 D: 0.04 I: 0.15
[/robot_1/module]: (1.2,5.8) with C:0.15 D: 0.35 I: 0.11
```

```
/home/yanling/api_ws/src/fyp_api/launch/performance_evaluation.launch http://localhost:11311 76
Time: 14.075     0.262----0.518----0.451----0.744
Receviled Shutdown: 0
Time: 14.763     0.262----0.668----0.509----0.872
Receviled Shutdown: 0
Time: 14.987     0.262----0.820----0.645----1.016
Receviled Shutdown: 0
Time: 15.628     0.263----0.968----0.792----1.162
Receviled Shutdown: 0
Time: 16.048     0.263----1.118----0.941----1.311
Receviled Shutdown: 0
Time: 16.657     0.263----1.267----1.090----1.461
```

```
yanling@yanling-Swift-SF314-42: ~/api_ws 76x12
yanling@yanling-Swift-SF314-42:~/api_ws$ ./save.sh
```

Figure G7: Exploration Run

Step 6: When the exploration is finished (as indicated by the evaluator in the third terminal) or earlier termination is wanted, run this command in the last terminal: ./save.sh



```
yanling@yanling-Swift-SF314-42: ~/api_ws 76x12
blocks_near__r2.yaml
[ INFO] [1647957089.216687696, 19.123000000]: Done

[ INFO] [1647957089.967366432]: Waiting for the map
[ INFO] [1647957090.188013663]: Received a 352 X 480 map @ 0.100 m/pix
[ INFO] [1647957090.188050891]: Writing map occupancy data to blocks_near__r
3.pgm
[ INFO] [1647957090.191140542, 19.533000000]: Writing map occupancy data to
blocks_near__r3.yaml
[ INFO] [1647957090.191226383, 19.533000000]: Done

/home/yanling/api_ws/src/fvp_api/launch/performance_evaluation.launch http://localhost:11311 76x
```

Figure G8: Screenshot after ./save.sh is run shows occupancy grids saved

./save.sh runs two processes for data collection.

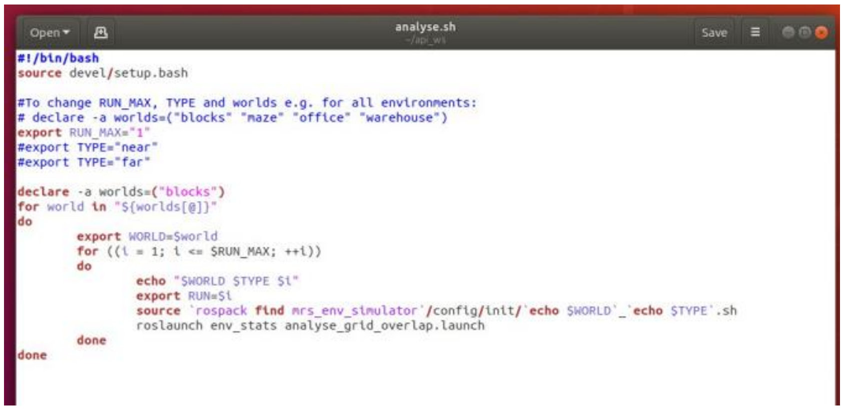
First, it takes a screenshot of the trajectory and saves the individual maps and merged map through *map_server* package for analysis. The (4) maps are saved as .pgm and .yaml files in the ~/(catkin_ws)/store directory. Recall that .yaml describes properties of the environment while .pgm contains the layout of the environment. This directory can be altered in the save.sh script. The filenames, however, follow this format: [WORLD]_[TYPE]_[CODE]_[RUN].pgm/.yaml, where WORLD, TYPE, CODE, and RUN are defined in params.sh file. For example, blocks_near_1_ is the first simulation in the Blocks using CFE.

Second, it sends a terminating condition to the *evaluator* node for output into a .txt file (e.g., stats.txt). If the exploration was terminated before completion without save.sh, no information will be collected. The .txt file is found in the data folder in the *cfe_module* package. The name for the .txt file as well as the directory can be edited in the performance_evaluator.launch file found in *cfe_module* package.

Post Simulation (Performance Analysis)

At this point, the exploration has concluded. The following steps are focused on the analysis of the maps related file (.pgm and .yaml) and the .txt file.

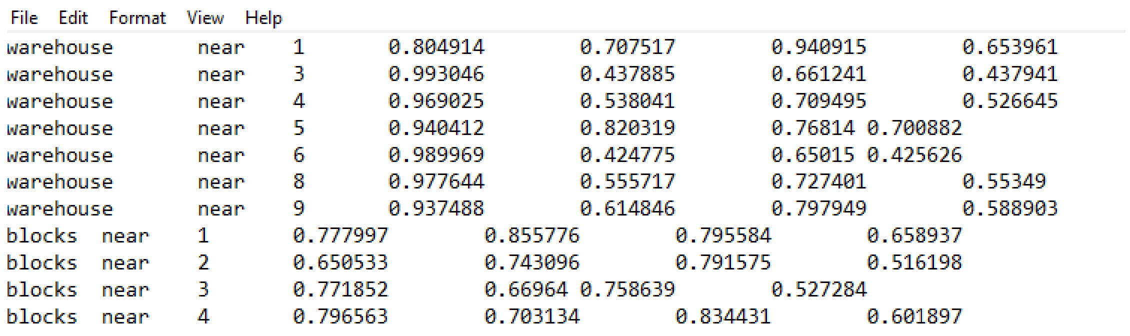
Step 7: To calculate the map overlap, run this command: ./analyse.sh



```
#!/bin/bash
source devel/setup.bash

#To change RUN_MAX, TYPE and worlds e.g. for all environments:
# declare -a worlds=("blocks" "maze" "office" "warehouse")
export RUN_MAX="1"
#export TYPE="near"
#export TYPE="far"

declare -a worlds=("blocks")
for world in "${worlds[@]}"
do
        export WORLD=$world
        for ((i = 1; i <= $RUN_MAX; ++i))
        do
                echo "$WORLD $TYPE $i"
                export RUN=$i
                source `rospack find mrs_env_simulator`/config/init/`echo $WORLD`_`echo $TYPE`.sh
                roslaunch env_stats analyse_grid_overlap.launch
        done
done
```

Figure G9: analyse.sh

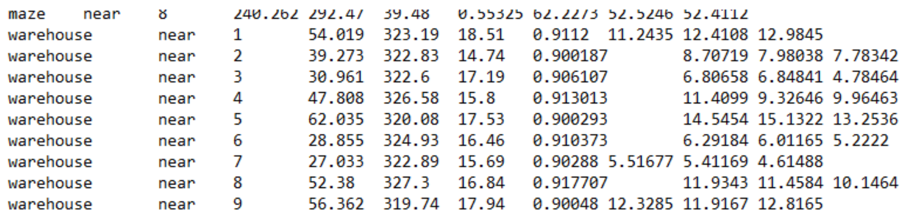
The standardisation of the filename automates the process of analysis, specifically, the calculation of map overlap. Ensure that the env variables RUN_MAX, TYPE, and worlds are properly set. The map overlaps will be recorded in a .txt file (e.g., overlaps.txt) in the /config folder of *env_stats* package. The values are reported in the order of: environment, type, run number, overlap in robot 1's map, robot 2's map robot 3's map, and the merged map as seen in Figure G10.



```
warehouse       near    1       0.804914        0.707517        0.940915        0.653961
warehouse       near    3       0.993046        0.437885        0.661241        0.437941
warehouse       near    4       0.969025        0.538041        0.709495        0.526645
warehouse       near    5       0.940412        0.820319        0.76814 0.700882
warehouse       near    6       0.989969        0.424775        0.65015 0.425626
warehouse       near    8       0.977644        0.555717        0.727401        0.55349
warehouse       near    9       0.937488        0.614846        0.797949        0.588903
blocks  near    1       0.777997        0.855776        0.795584        0.658937
blocks  near    2       0.650533        0.743096        0.791575        0.516198
blocks  near    3       0.771852        0.66964 0.758639        0.527284
blocks  near    4       0.796563        0.703134        0.834431        0.601897
```

Figure G10: Sample overlaps.txt file

Step 8: To access the .txt file collected by *evaluator* node (stats.txt), navigate to */data* folder in the *cfe_module* package. The values are reported in the order of: environment, type, run number, exploration time, free areas, occupied areas, explored ratio, r1 dist, r2 dist, r3 dist as seen in Figure G11.



```
maze    near    8       240.262 292.47  39.48   0.55325 62.2273 52.5246 52.4112
warehouse       near    1       54.019  323.19  18.51   0.9112  11.2435 12.4108 12.9845
warehouse       near    2       39.273  322.83  14.74   0.900187        8.70719 7.98038 7.78342
warehouse       near    3       30.961  322.6   17.19   0.906107        6.80658 6.84841 4.78464
warehouse       near    4       47.808  326.58  15.8    0.913013        11.4099 9.32646 9.96463
warehouse       near    5       62.035  320.08  17.53   0.900293        14.5454 15.1322 13.2536
warehouse       near    6       28.855  324.93  16.46   0.910373        6.29184 6.01165 5.2222
warehouse       near    7       27.033  322.89  15.69   0.90288 5.51677 5.41169 4.61488
warehouse       near    8       52.38   327.3   16.84   0.917707        11.9343 11.4584 10.1464
warehouse       near    9       56.362  319.74  17.94   0.90048 12.3285 11.9167 12.8165
```

Figure G11: Sample stats.txt file

The values are separated by tabs and can be pasted into excel for analysis.

Appendix H: Documentation for *mrs_env_simulator*

This package spawns multiple robots in a desired Gazebo Environment (tested on ROS1 Melodic 18.04). Refer to Figure G1 for recommended path for this package.

Requirements:

- move_base package: sudo apt-get install ros-melodic-navigation
- gmapping: sudo apt install ros-melodic-slam-gmapping
- kobuki: sudo apt-get install ros-melodic-kobuki ros-melodic-kobuki-core
- fyp_commands (contains all the .sh scripts needed)

Package Structure:

- /config: contains the configuration for initial positions (/init), parameters used for SLAM and navigation (/param), and rviz visualiser window (/rviz)
- /gazebo: contains the gazebo worlds and models. Place the created files from *map2gazebo* (see Appendix A) here
- /maps: contains the ground truth map for the 4 environments. Used for visualisation purposes only.
- /launch: contains launch files to launch the required nodes.

Usage:

```
~/catkin_ws: source devel/setup.bash
~/catkin_ws: source (path_to_package)/config/init/[WORLD]_[TYPE].sh
      e.g., source src/mrs_env_simulator/config/init/blocks_far.sh
~/catkin_ws: roslaunch mrs_env_simulator env_three.launch
```

Alternatively,

```
~/catkin_ws: ./env.sh
```

Table H1: Topics related to *mrs_env_simulator package*

Published Topics:	Subscribed Topics:
• odom (nav_msgs/Odometry) • map (nav_msgs/OccupancyGrid) • tf (tf/tfMessage)	• move_base_simple/goal (geometry_msgs/PoseStamped)

Appendix I: Documentation for *cfe_module*

This package runs the proposed algorithm for exploration: Coordinated Frontier Exploration (tested on ROS1 Melodic 18.04). Refer to Figure G1 for recommended path for this package.

Requirements:

- move_base package: sudo apt-get install ros-melodic-navigation

Package Structure:

- /config: contains the configuration for ROS parameters used for CFE (see Appendix F) and path planner module.
- /msg: contains custom ROS messages created for exchange of information between robots
- /data: folder to store the performance of exploration
- /launch: contains launch files to launch the required nodes.
    - For robot team size of n, these nodes will be spawned:
        - n *exploration_module* nodes
            - n *detector* nodes
            - n *allocator* nodes
        - 1 *filter* node

Nodes:

- Detector. The following source codes form this node:
    - Exploration_module.cpp: Highest Level, contains main function that executes at pre-defined frequency
    - Planner.cpp: Sends and receives information from nodes in *mrs_env_simulator* or other *exploration_module* nodes belonging to other robots
    - Robot.cpp: Tracks of own statistics (e.g., pose, frontier candidates) and the team's statistics and update the environment accordingly.

    - Environment.cpp: Main functionality for detector. With information on robot and team, detect frontiers, cluster frontiers, and allocate.
    - Visualisation.cpp: ease of visualising vectors of points
    - Functions.cpp: contains misc functions used throughout the files
- Filter. The following source code form this node:
    - Assigner.cpp: Receives all generated centroid and outputs a filtered list of centroids through checks and clustering
- Allocator: The following source code form this node:
    - Environment.cpp
- Evaluator:
    - performance_evaluation.cpp: Subscribes to individual robot's odometry and map and publish metrics (to both terminals and file).

Usage:

```
~/catkin_ws: source devel/setup.bash
~/catkin_ws: roslaunch cfe_module 2d_exploration.launch
~/catkin_ws: roslaunch cfe_module performance_evaluation.launch
```

Alternatively,

```
~/catkin_ws: ./run.sh
~/catkin_ws: ./stats.sh
```

Table I1: Topics related to *exploration_module* node

Published Topics:	Subscribed Topics:
• move_base_simple/goal (geometry_msgs/PoseStamped)	• odom (nav_msgs/Odometry) • map (nav_msgs/OccupancyGrid) • tf (tf/tfMessage) • /frontier/finished (std_msgs/Bool)
Published + Subscribed Topics	
• /frontier/centroids (cfe_module/centroidsArray) • /frontier/goal (geometry_msgs/PoseStamped	

Table I2: Topics related to *filter* node

Published Topics:	Subscribed Topics:
	• map (nav_msgs/OccupancyGrid)
Published + Subscribed Topics	
• /frontier/centroids (cfe_module/centroidsArray)	

Table I3: Topics related to *evaluator* node

Published Topics:	Subscribed Topics:
• traj (nav_msgs/Path) • merged_map (nav_mgs/OccupancyGrid) • /frontier/finished (std_msgs/Bool)	• odom (nav_msgs/Odometry) • map (nav_msgs/OccupancyGrid) • tf (tf/tfMessage)

Appendix J: Documentation for *env_stats*

This package provides analysis on statistics related to explorations. This package is used either before or after the exploration runs, but never during. For pre-simulation, the package offers calculation of total free and occupied areas in the environment. For post-simulation, which is its main functionality, it computes map overlap. This has been tested on ROS1 Melodic 18.04. Refer to Figure G1 for recommended path for this package.

Requirements:

- map_server: sudo apt-get install ros-melodic-map-server

Package Structure:

- /config: folder to store output files
- /data: contains the map related files (*.pgm and *.yam) between robots. This is where the maps will be retrieved, and it is important to ensure the files are located here.
- /launch: contains launch files to launch the required nodes.

For post-simulation analysis

Node:

- *map_analysis*
  - grid_overlap.cpp: see Section 3.2 (Map Overlap) for theory. see Appendix C for the main function source code. Calculated map overlap is found in (default name) overlaps.txt in /config folder.

Usage:

```
~/catkin_ws: source devel/setup.bash
~/catkin_ws: source params.sh
~/catkin_ws: roslaunch env_stats analyse_grid_overlap.launch
```

Alternatively,

```
~/catkin_ws: ./analyse.sh
```

Table J1: Topics related to *map_analysis* node

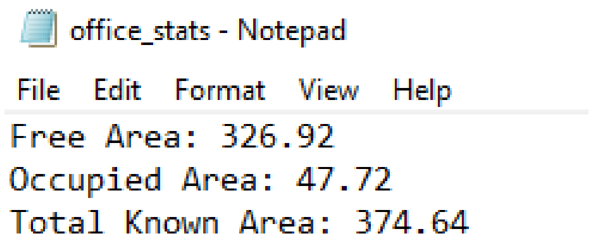
Published Topics:	Subscribed Topics:
• map_shifted (nav_msgs/OccupancyGrid)	• map (nav_msgs/OccupancyGrid)

For robot team size of 3, 4 maps (3 individual maps + 1 merged map) will be provided as input.

<u>For pre-simulation analysis</u>

Node:

- *env_stats_generator*
    - env_stats.cpp: Iterates through an occupancy grid to give total free area (value = 1) and occupied area (value = -1). Values are printed onto (default name) office_stats.yaml. This is particularly useful if the Gazebo world is created by user as information on the total known area is needed to quantify exploration efforts.



```
Free Area: 326.92
Occupied Area: 47.72
Total Known Area: 374.64
```

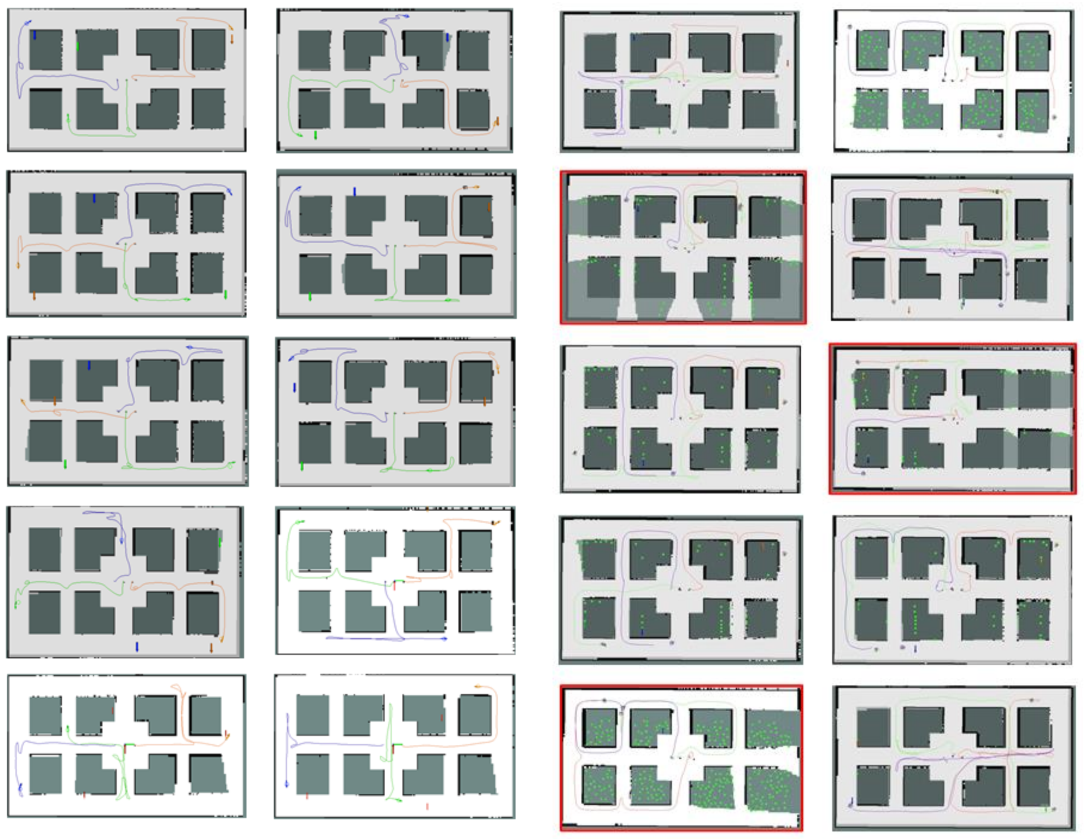
Figure J1: Sample office_stats.yaml file

Usage:

```
~/catkin_ws: source devel/setup.bash
~/catkin_ws: roslaunch env_stats env_stats_generator.launch
```

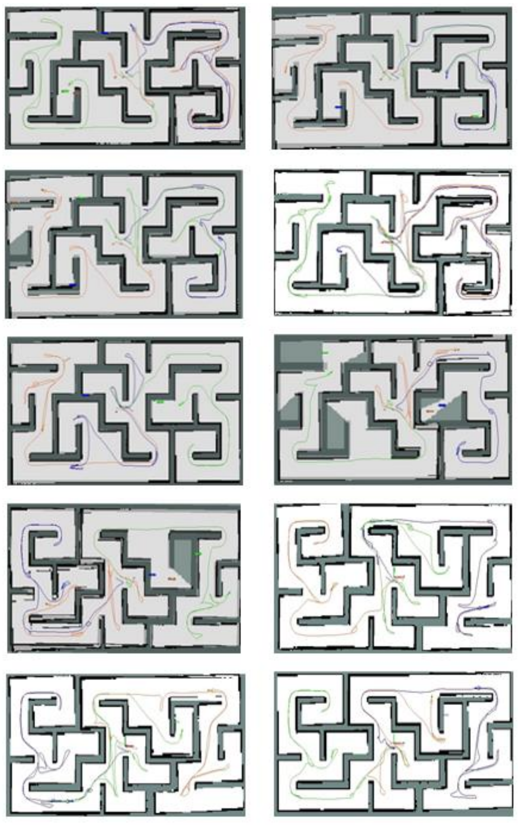
Appendix K: Trajectories for All Simulation

Figure K1-K4: CFE vs RRT in Blocks, Maze, Office, and Warehouse, respectively. Runs that are outlined in red are incomplete.



(a) CFE (b) RRT

Figure K1: Comparison of CFE and RRT in Blocks


(a) CFE

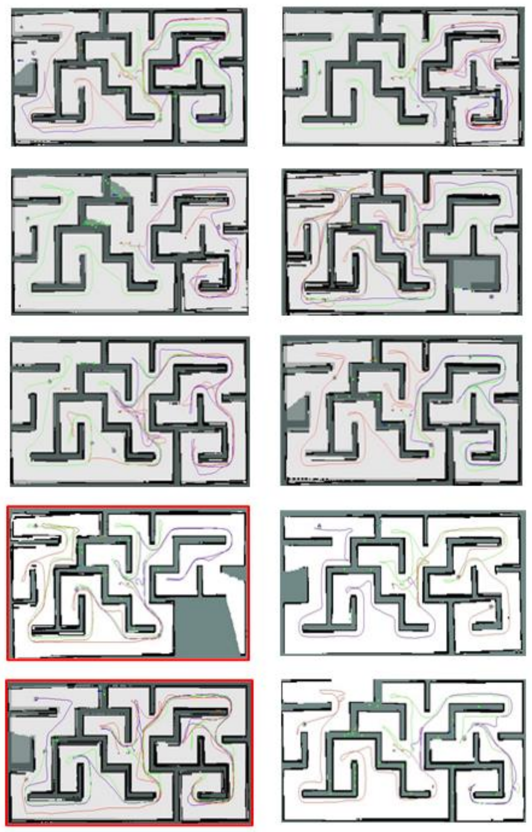
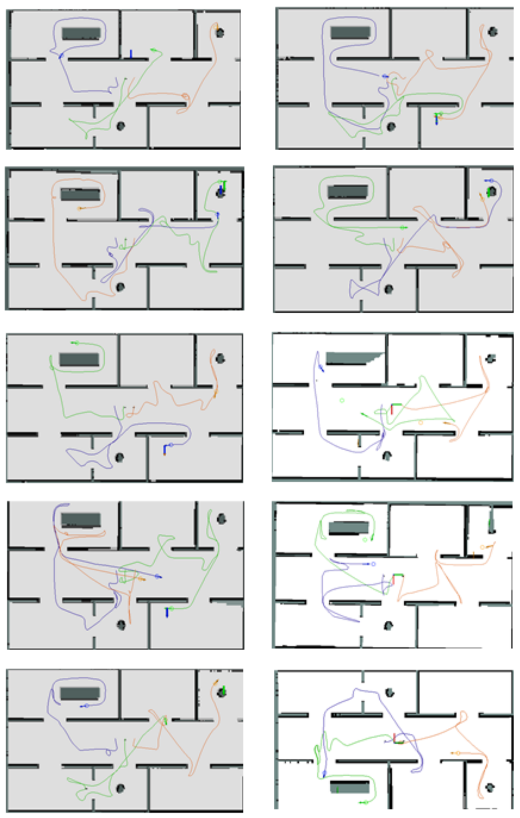

(b) RRT

Figure K2: Comparison of CFE and RRT in Maze


(a) CFE

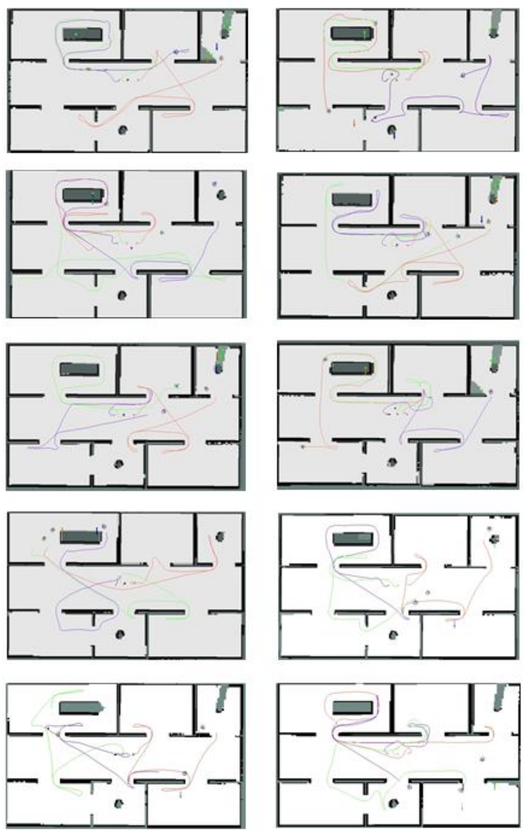
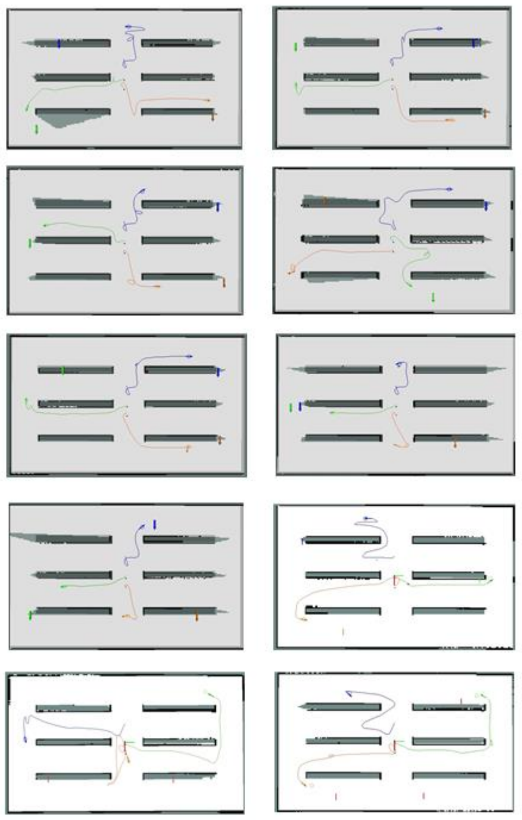

(b) RRT

Figure K3: Comparison of CFE and RRT in Office


(a) CFE

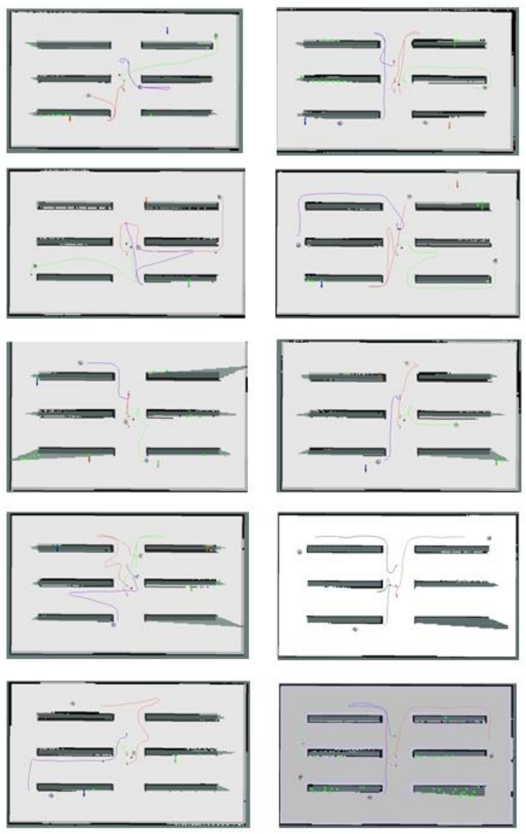

(b) RRT

Figure K4: Comparison of CFE and RRT in Warehouse