POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Autonomous Robot Exploration using Deep Learning: An Experimental Analysis

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Marco Premi

Advisor: Prof. Francesco Amigoni

Co-advisor: Dr. Matteo Luperto

Academic year: 2021-2022

1. Introduction

Among the different research fields in robotics, autonomous mobile robotics has been actively addressed in the last years. Autonomous *exploration* is one of the most important tasks that an autonomous robot, deployed in an unknown environment, must accomplish. The robot, with no previous information about the environment, has to choose where to move and consequently which is the best strategy to explore the environment in order to build its map incrementally. Over the years, different strategies have been proposed and developed. Even if these classical techniques proved to be mostly successful, a recent research thrust aims to develop Machine Learning, and in particular Deep Learning, techniques to address the exploration problem, because of the performance achieved by these approaches in other fields. Deep Learning methods require a lot of data and, in most cases, when compared to classical methods they lack explainability. In addition, most of the Deep Learning exploration algorithms (known to exploit imperfections of the simulators) are studied only in simulation environments and only sometimes have also been tested in real-world environments (contrary to classical algorithms). The purpose of this thesis is to experimentally compare classical and Deep Learning algorithms for exploration in order to understand which are the positive and negative sides of the different techniques. This comparison can provide interesting results because of the fact that the community of researchers that is dedicated to study classical exploration algorithms and the one that is dedicated to study Deep Learning exploration paradigm are in most of the cases separate communities. Many researchers in the field of classical exploration rely on classical techniques as they have proven to be robust and with reliable performance also when used by actual robots in the real world; however, this focus on classical techniques may prevent them to benefit from innovations and better performance that may be offered by Machine Learning and in particular Deep Learning. Deep Learning researchers, on the other hand, in most of the cases compare their algorithms only with other learning-based approaches, without taking into account classical exploration algorithms and the challenge of deploying the agents in the real world.

2. State of the art

In order to understand how exploration tasks can be solved and where classical and Deep Learning techniques can be used, we have identified from the literature three macro modules, as shown in Figure 1.

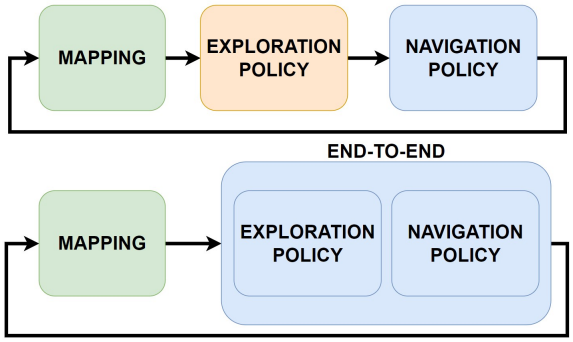

Figure 1: Schematization of the exploration problem. On the top, the case where exploration and navigation policies are considered separate modules. On the bottom, the case where exploration and navigation policies are considered a single module, learned end-to-end.

Mapping is the module in charge of generating or updating the map. The map can be simply updated with the information coming from the current observations or enhanced with predictions. *Exploration policy* module is responsible for the selection of the next point that must be reached during the exploration, while *navigation policy* is the module responsible for reaching that point. Exploration and navigation can also be solved end-to-end with Deep Learning algorithms.

Mapping with classical algorithm is usually obtained with *SLAM (Simultaneous Localization and Mapping)* [16]. With SLAM, we indicate the task of building the map of the environment while trying to localize the agent in it: the algorithm takes as input odometry and sensors data and outputs the updated map and robot location. One example of mapping with Deep Learning is provided in [11] where authors use a U-net network that takes as input an rgb image and produces as output an egocentric top-down 2D spatial map.

Most of the classical methods proposed to select the next exploration location and therefore implement the exploration policy module are based on the concept of *frontiers*, defined as the regions between free and unexplored space. The most famous algorithm is called *frontier exploration* [18]. With this approach, the robot moves to the closest frontier, until no more frontiers are reachable. The next frontier to be visited can be selected in multiple ways, i.e., randomly or by selecting the frontier point that maximizes the amount of unknown area that can be seen from it. In the literature, different Deep Learning algorithms have been proposed to implement the exploration policy. One example is provided in [11], where authors use a Convolutional Neural Network (CNN) which takes as input the updated map and the pose estimate and outputs a long-term goal. The CNN is trained with Reinforcement Learning and the reward is proportional to the increase in area coverage. Subsequently, they compute the shortest path between the current position and the long-term goal in order to select a short-term goal along the path. In the literature, other works follow this structure, proposing algorithms that reward other aspects of the exploration. A positive reward can be given, for example, if actions are smooth or if the goal is reached fast. On the other hand, for example, a negative reward can be given if a collision occurs.

In the literature, many exploration algorithms use classical techniques like A* or Dijkstra to plan the path from the current position to the point selected by the exploration policy and therefore implement the navigation policy module, because these algorithms already offer the optimal solution. Several works use Deep Learning to learn a navigation policy in order to reach a point goal selected by the exploration policy previously defined. In almost all the literature cases we have examined, Deep Learning policies are trained with Reinforcement Learning. One example is the one reported in [12], where authors use an actor-critic network trained with Reinforcement Learning which takes as input the description of the local environment (bagged laser readings in 180 degrees range in front of the robot) and the waypoint goal that the agent must reach and outputs the next action that must be executed by the robot. The policy gives a positive reward if the distance to the goal is less than a threshold, a negative reward in case of collision and if none of the previous conditions are satisfied it gives a

reward based on the current linear velocity and angular velocity.
The last case we have to consider is the one where exploration and navigation are jointly learned using an end-to-end paradigm and can't be clearly separated into two modules as in the previous cases. One example is provided by authors of [13]. The authors compare different Reinforcement Learning algorithms that output the direction that must be followed by the agent. The input is represented by the sensory information acquired by the robot in the environment. All the algorithms are trained with the same reward function that gives a positive reward if new cells are discovered, a negative reward per timestep (in order to avoid unnecessary movements), a negative reward if the action done leads to invalid states, and a positive reward when more than a threshold value of the cells have been explored.

3. Algorithms selected for comparison

In this section, we describe the classical and Deep Learning algorithms we have selected from the literature in order to perform the comparison and understand what are the downsides and upsides aspects of the different implementations, taking into consideration the different modules identified in Section 2.

3.1. Classical exploration algorithm

We have selected as an example of a classical method the greedy frontier exploration of [4]. In greedy frontier exploration, the robot chooses as the next exploration goal the point, from the frontier list, closest to the actual position. The classical algorithm used for comparison in this thesis is simple, but other more complex and efficient algorithms exist in the literature.

- **Input:** laser readings and odometry readings.
- **Mapping module:** SLAM (implemented with Gmapping [6]), the module outputs a grid-based map.
- **Exploration policy module:** greedy frontier exploration (implemented with explore_lite package of ROS [4]), the module outputs the frontier point to be reached.
- **Navigation policy module:** Dijkstra for path selection (implemented with move_base package of ROS [7]), the module outputs the stream of velocity commands the robot has to execute to reach the selected point.

3.2. Deep Learning algorithms

In this section, we describe the Deep Learning algorithms selected to perform the comparison: ANS from [11], OccAnt [14], and DRL [12].

3.2.1 Learning to explore using active neural SLAM - ANS

- **Input:** sensor pose readings and observations (rgb images).
- **Mapping module:** *Neural SLAM* module. This module takes as input the current rgb observation, the current and last sensor reading of the agent pose, the last map estimate, and the last agent pose. It outputs the updated grid-based map and the current agent pose estimate.
- **Exploration policy module:** a CNN trained with Reinforcement Learning takes as input the map, the pose estimated by the Neural SLAM module, and the visited locations and outputs the long-term goal. The policy rewards large area coverage. Then, a planner takes as input the long-term goal, the obstacle map, and the agent pose to output the short-term goal using the Fast Marching method [15].
- **Navigation policy module:** a Recurrent Neural Network (RNN). It takes as input the observations (rgb images) and the short-term goal produced by the exploration policy module. The policy is trained using Imitation Learning to output a navigational action.

3.2.2 Occupancy anticipation for efficient exploration and navigation - OccAnt

- **Input:** rgb images, depth images, and sensor pose reading.
- **Mapping module:** a U-net network that takes as input rgb and depth images and outputs the anticipated occupancy (occupancy of areas that can't be directly seen) for the region in front of the agent.

- **Exploration policy module:** the same network as that of Section 3.2.1 with a small difference; while ANS rewards increase in area coverage, OccAnt rewards actions that allow to predict correctly the map, irrespective of the fact that the robot has actually observed that predicted location or not.
- **Navigation policy module:** the same as Section 3.2.1.

3.2.3 Goal-driven autonomous exploration through Deep Reinforcement Learning - DRL

- **Input:** laser readings and odometry readings.
- **Mapping module:** classical SLAM, the module outputs a grid-based map.
- **Exploration policy module:** the algorithm at each step selects a list of *POIs* (Points of Interest) with a numerical method. The optimal POI, provided as module's output, is selected with an evaluation method developed by the authors of the algorithm, *Information-Based Distance Exploration (IDLE)*.
- **Navigation policy module:** an actor-critic network trained with Reinforcement Learning which takes as input the description of the local environment and the waypoint goal that the agent must reach and outputs the next action that must be executed by the robot. The policy gives a positive reward if distance to the goal is less than a threshold, a negative reward in case of collision and, if none of the previous conditions are satisfied, it gives a reward based on the current linear velocity and angular velocity.

4. Experimental comparison

In this section, we describe the experimental setting of the comparison between classical and Deep Learning algorithms and its results.

4.1. Experimental setting

ANS and OccAnt are tested with the original code provided by authors in [8], with authors' pretrained models. OccAnt and ANS code implementation are strongly dependent on the tools provided by the Habitat simulator [1] and because of this they can't be tested on other simulation environments. As a consequence, we rely on the Habitat simulator as the main simulator used in the comparison. On the other hand, DRL (tested with the code provided by authors in [3]) and frontier exploration algorithm (with our own implementation) are originally implemented using ROS [9]. We test them in the Habitat simulator using ROS-X-Habitat [10], a software interface able to bridge the Habitat simulator with other robotics resources which use ROS. Habitat allows algorithms to be tested with different photorealistic datasets. All the photorealistic environments used during the comparison are assumed to be static, meaning that in the environments no any other agent is moving and the environment itself doesn't change during the exploration. The environments used are taken from the Gibson dataset, a reproduction of real indoor spaces made with 3D scanning and reconstructions. Frontier exploration, for a reason that emerged during the execution of the comparison, is also tested in the Stage simulator [17] in 2D versions of the same environments (better explained in Section 4.2.2). Algorithms are compared on two different exploration tasks: exploration for map building and point-goal driven exploration. In exploration for map building the agent is required to explore the environment and build a map as accurate as possible, while in point-goal driven exploration the agent is required to explore the environment and build a map used to reach the point provided as input. In exploration for map building task comparison, we change the starting location in the environment at every episode. In point-goal driven exploration task, we change starting and goal locations in the environment at every episode, and a goal is considered reached if the agent's distance to it is lower than 0.3 m. OccAnt and ANS present a very similar structure and because of this, we run comparison episodes between them in order to understand if the introduction of the occupancy anticipation in the mapping module allows OccAnt to have better performance compared to ANS in both the tasks. OccAnt, ANS, and the classical algorithm are compared on both tasks, while the DRL algorithm only on point-goal driven exploration task. DRL is originally trained and tested in very simple Gazebo

[5] environments (only a small room with a small number of furniture elements) and because of this, we run some test episodes in more complex Habitat environments in order to understand if the algorithm is able to generalize well in unknown environments. ANS and OccAnt can be tested with different inputs: rgb, depth, or rgbd. In the following tables, we only report the implementation of ANS and OccAnt with the best results and not all the implementations with all the different inputs. The two tasks are compared with task-specific metrics. Some of them are self-explanatory, others must be specificied. In exploration for map building task with map_accuracy we refer to the area of the global map built during exploration that matches the ground-truth provided, with AC/AS we refer to the ratio between map_accuracy and area_seen (the more the value is close to one, the more the area is mapped accurately). In point-goal driven exploration task with dist_to_goal, we refer to the distance to the goal (in m) at the end of the episode (the lower the value, the closest every episode ends to the goal), with num_step we refer to the total number of steps taken during the episode in Deep Learning algorithms.

4.2. Experimental results

In this section, we describe all the tests done to compare classical and Deep Learning algorithms in the two exploration tasks.

4.2.1 OccAnt vs. ANS - Exploration for map building task

	ANS (depth)	OccAnt (rgb)
map_accuracy (m^2)	47.451 (17.449)	**48.725 (18.142)**
area_seen (m^2)	55.609 (19.849)	**56.913 (21.031)**
free_space_seen (m^2)	29.801 (11.002)	**30.991 (12.010)**
occupied_space_seen (m^2)	25.808 (9.437)	**25.922 (9.415)**
time_per_episode (s)	**0.865 (0.002)**	0.984 (0.004)
area_seen_over_time (m^2/s)	**64.288 (22.242)**	57.838 (20.072)
AC/AS	0.853 (0.045)	**0.856 (0.042)**

Table 1: Results of comparison on exploration for map building done with OccAnt and ANS on 994 episodes in 14 environments of different sizes (in parentheses, we report the standard deviation).

Looking at the values in Table 1 we can see that neither algorithm is significantly better than the other. OccAnt (rgb) proved to be the best choice for different metrics (map_accuracy, area_seen, free_space_seen, and occupied_space_seen) but looking at area_seen_over_time, ANS (depth) is able to explore more area in the same time. In any case, with higher average values come also higher standard deviation values. The presence of occupancy anticipation module in OccAnt (rgb) doesn't give to OccAnt (rgb) a clear advantage in exploration for map building task, but in point-goal driven exploration task (as later seen in Section 4.2.3) it allows better performance.

4.2.2 OccAnt vs. ANS vs. frontier exploration - Exploration for map building task

In this section, we show the results of the comparison done between frontier exploration (in 2D, Stage simulator, and 3D, Habitat simulator), OccAnt, and ANS (both in Habitat simulator). For performance reasons, we have tested the three algorithms only on some maps, but with significant differences in size. Frontier exploration algorithm uses laser readings as input (while OccAnt and ANS use rgb or depth images) and sometimes these readings in the Habitat simulator with Gibson dataset don't fully represent a real world situation. The textures present holes and so laser readings go beyond walls or objects creating frontiers that don't exist in the real world. With ROS-X-Habitat implementation, in all the episodes analyzed, the frontier exploration agent gets stuck after some time, due to some inaccurate textures and it can't move anymore. In order to propose a more qualitative analysis of the frontier exploration algorithm and discover what happens when the agent doesn't get stuck, we have also tested the frontier exploration algorithm in 2D maps in the Stage simulator.
First of all, we have compared the maps obtained during exploration by OccAnt, ANS, and frontier exploration (3D and 2D) with a fixed path_length (the one when frontier exploration 3D gets stuck). One example is reported in Figure 2.

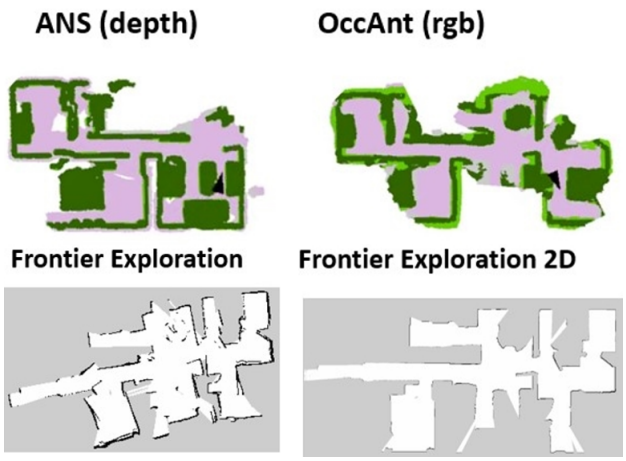

Figure 2: Map produced by ANS (depth), OccAnt (rgb), and frontier exploration (2D and 3D).

From the figure, which describes a situation similar to what happens also in the other exploration episodes, we can see that all the algorithms explore a comparable amount of the environment. For this reason, we can use frontier exploration 2D instead of frontier exploration 3D in order to have a more qualitative analysis, taking into consideration free_space_seen metric. It would not be fair to consider free_space_seen metric with frontier exploration 3D because the holes in the textures already mentioned allow to see areas that can't be seen in a real-world situation.

	free_space_seen (m^2)
ANS(depth)	28.17 (9.091)
OccAnt(rgb)	30.50 (11.326)
Frontier exploration 2D	31.39 (12.311)

Table 2: Average of the free_space_seen with the same path_length in OccAnt, ANS, and frontier exploration 2D over environments of different dimensions (in parentheses, we report the standard deviation).

From Table 2 we can see that in these environments there is no substantial advantage in using Deep Learning techniques over classical well-established techniques. In order to fully compare OccAnt, ANS, and frontier exploration we have also compared the decision-making time required by each algorithm to select the next action to be executed at each exploration step. What comes out is that the decision-making time in OccAnt and ANS is lower than the one of frontier exploration: for ANS is 0.001722 s, for OccAnt is 0.001958 s, and for frontier exploration is 0.095430 s. However, frontier exploration decision-making time is still low and fully compatible with use in the real world.

4.2.3 OccAnt vs. ANS - Point-goal driven exploration task

	ANS (depth)	OccAnt (depth)
succes_rate	0.722	**0.867**
dist_to_goal (m^2)	1.911 (3.448)	**1.030 (2.028)**
num_steps	208.792	**145.561**

Table 3: Results of comparison on point-goal driven exploration done with OccAnt and ANS on 994 episodes in 14 environments of different sizes (in parentheses, we report the standard deviation).

Table 3 provides us with the information that OccAnt (depth) obtains sensibly better results than the best ANS configuration i.e., ANS (depth). It is interesting to note that while in exploration for map building task OccAnt (rgb) and ANS (depth) obtain very similar results in all the metrics (Section 4.2.1), in point-goal driven exploration task, OccAnt (depth) obtains noticeable better results thanks to the presence of the occupancy anticipation module. In point-goal driven exploration task, contrary to what happens in exploration for map building task, with higher average values comes lower standard deviation values.

4.2.4 DRL tests - Point-goal driven exploration task

In order to understand how the DRL algorithm works and how it can be compared with ANS or OccAnt algorithms, we have run different tests. First of all we have run a training for the DRL algorithm in the original Gazebo simple environments. This training lasted about 12 hours and is enough to obtain 100% success_rate during evaluation in Gazebo environments. Then, we have tested the algorithm in the Habitat complex environments used for ANS and OccAnt comparison and results are sensibly worse (success_rate is reduced to 0.26%). Analyzing the start and goal positions in case of success it appears that the agent is able to reach the goal only when start and goal positions are very close (probably in the same room), being the set-up similar to the one in the DRL simple environments.

In another test, we have trained the DRL algorithm for 317 episodes in the Habitat simulator. The robot is now able to cover more space than the robot trained in the more simple Gazebo environments, but always with low values in success_rate (0.25%). A strange behavior appears when we test these weights learned in the Habitat simulator back in Gazebo DRL environment: the robot moves in circle and doesn't try to reach the goal. One possible explanation is the fact that in Habitat environments there are a lot of obstacles and consequently a lot of collisions. Because of the negative reward obtained with collisions, the robot has learned a policy that prefers not colliding over reaching the goal. We have come to this conclusion also by looking at the training videos. In the first episodes, the robot collides a lot trying to reach the goal, but then in the subsequent episodes, it collides much less.

4.2.5 OccAnt vs. ANS vs. classical algorithm - Point-goal driven exploration

In this section, we show the results of the point-goal driven exploration task comparison between OccAnt, ANS, and the classical algorithm (implemented with ROS Gmapping [6] and move_base [7]) on 90 episodes on 3 environments of different sizes (sorted in ascending order by area). ANS and OccAnt are tested in the Habitat simulator, while the classical algorithm, for the reasons described in Section 4.2.2 is tested in the Stage simulator.

	classical	OccAnt (depth)	ANS (depth)
Env. #1	**21**	3	6
Env. #2	**19**	7	4
Env. #3	**19**	10	1
Total	**59**	20	11

Table 4: Number of best path_length episodes in classical algorithm, OccAnt (depth), and ANS (depth).

	classical	OccAnt (depth)	ANS (depth)
Env. #1	**0.967**	0.900	0.800
Env. #2	**0.967**	0.900	0.767
Env. #3	**0.967**	0.833	0.600
Total	**0.967**	0.878	0.722

Table 5: success_rate comparison between classical algorithm, OccAnt (depth), and ANS (depth).

Results of Table 4 show that the classical algorithm, no matter the size of the environment, performs better: it has the best path_length in most of the episodes. In every environment, the classical algorithm also has more success episodes (as shown in Table 5). Even if OccAnt (depth) with occupancy anticipation module proved to perform better with respect to ANS (depth) (Section 4.2.3), there is no clear advantage over the classical algorithm neither in success_rate nor in path_length metric.

5. Conclusion

In this thesis, we have compared classical and Deep Learning algorithms in common environments on two different tasks: exploration for map building and point-goal driven exploration, in order to understand what are the downsides and upsides of the algorithms proposed.
The comparison done between OccAnt and ANS shows that the introduction of the occupancy anticipation module helps OccAnt to get better results only in point-goal driven exploration task.
The comparison done between OccAnt, ANS, and frontier exploration in exploration for map building shows that the three algorithms obtain comparable results. OccAnt and ANS, however, require a very long training time with big computational power and energy resources involved [2]. On the other side, frontier exploration can be simply implemented on an agent and doesn't have to deal with all the issues regarding training or ability to generalize to an unknown environment, because its performance doesn't depend on the number and the quality of training environments. In addition, frontier exploration algorithm already proved to work well when tested in real world environments. Frontier exploration decision-making time, even if higher than the one of OccAnt and ANS is fully compatible with real world situations.
The tests done on DRL algorithm highlights the difficulty of writing a strong reward function, which is also able to generalize well in unknown environments. This element of difficulty is not present using classical algorithms and should be carefully considered.
From the comparison between OccAnt, ANS, and the classical algorithm on point-goal driven exploration task we can see that also in this task the classical algorithm obtains comparable

or better results with respect to Deep Learning algorithms.
Deep Learning algorithms are known to exploit imperfections in order to obtain better performance and because of this fact, in the future, it would be useful to know if these results can also be achieved in the real world in complex environments and not only in simple environments like the one we have described for DRL algorithm.

References

[1] AI Habitat. `https://aihabitat.org/`. Last accessed: 06/01/2022.

[2] Ans training time. `https://github.com/devendrachaplot/Neural-SLAM/issues/30`. Last accessed: 18/04/2022.

[3] Drl robot navigation. `https://github.com/reiniscimurs/DRL-robot-navigation`. Last accessed: 06/01/2022.

[4] Explore lite - frontier exploration. `http://wiki.ros.org/explore_lite`. Last accessed: 01/02/2022.

[5] Gazebo simulator. `http://gazebosim.org/`. Last accessed: 16/01/2022.

[6] Gmapping - SLAM. `http://wiki.ros.org/gmapping`. Last accessed: 01/02/2022.

[7] movebase. `http://wiki.ros.org/move_base`. Last accessed: 14/03/2022.

[8] Occupancy anticipation. `https://github.com/facebookresearch/OccupancyAnticipation`. Last accessed: 16/01/2022.

[9] ROS. `https://ros.org`. Last accessed: 07/01/2022.

[10] ROS-X-Habitat. `https://github.com/ericchen321/ros_x_habitat`. Last accessed: 16/01/2022.

[11] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. *arXiv preprint arXiv:2004.05155*, 2020.

[12] Reinis Cimurs, Il Hong Suh, and Jin Han Lee. Goal-driven autonomous exploration through deep reinforcement learning. *IEEE Robotics and Automation Letters*, 7(2):730–737, 2022.

[13] Dimitrios I Koutras, Athanasios Ch Kapoutsis, Angelos A Amanatiadis, and Elias B Kosmatopoulos. Marsexplorer: Exploration of unknown terrains via deep reinforcement learning and procedurally generated environments. *arXiv preprint arXiv:2107.09996*, 2021.

[14] Santhosh K Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy anticipation for efficient exploration and navigation. In *Proceedings of the European Conference on Computer Vision*, pages 400–418, 2020.

[15] James A Sethian. Fast marching methods. *SIAM review*, 41(2):199–235, 1999.

[16] Cyrill Stachniss, John J Leonard, and Sebastian Thrun. Simultaneous localization and mapping. In *Springer Handbook of Robotics*, pages 1153–1176. Springer, 2016.

[17] Richard Vaughan. Massively multi-robot simulation in stage. *Swarm intelligence*, 2(2):189–208, 2008.

[18] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.*, pages 146–151, 1997.