Autonomous Frontier Exploration and Mapping with Applications in Search and Rescue

Dominic Garrity
John A. Paulson School of Engineering and Applied Sciences
Harvard University
Cambridge, Massachusetts

Arif Kerem Dayı
John A. Paulson School of Engineering and Applied Sciences
Harvard University
Cambridge, Massachusetts

Emeka Ezike
John A. Paulson School of Engineering and Applied Sciences
Harvard University
Cambridge, Massachusetts

Victor Gonçalves
John A. Paulson School of Engineering and Applied Sciences
Harvard University
Cambridge, Massachusetts

***Abstract*—Though robotics has made many aspects of day-to-day life easier, its applications in disaster response are still limited. Search-and-rescue missions, for example, are typically conducted by humans and often result in failure, injury, or death. While research has been done on supporting these missions with robots, it often assumes very little noise and error and is therefore difficult to apply. For this reason, we investigated whether and how relatively inexpensive robots–namely, LoCoBots–could autonomously explore and map novel indoor environments. We did so through a simulation study as well as hardware experiments. Our empirical findings suggested that LoCoBots are able to autonomously explore and map novel indoor environments. In addition, our findings reveal the extent to which the wavefront frontier detection algorithm is slower than the greedy algorithm on large simulated maps, but faster than the greedy algorithm on small real-world maps. This work not only covers promising exploration and mapping methods, but also details their limitations.**

***Index Terms*—Robotics, disaster response, search and rescue, frontier detection, path planning, SLAM**

I. Introduction

Year	*Total SAR cost**	*Total SAR operations*	*Fatalities*	*Illness or injured*	*Nonillness or injured*	*Save*
2007	$4 735 424	3593	136	1218	2566	1023
2006	$4 524 875	3623	119	1445	2900	1211
2005	$4 996 705	2430	152	1129	2016	402
2004	$3 592 218	3216	127	1087	3077	815
2003	$3 468 255	3108	124	1199	2162	427
2002	$3 040 020	4537	129	1338	3492	1832
2001	$3 683 086	3619	123	1502	2782	155
2000	$2 779 967	4869	244	1471	3495	709
1999	$3 483 500	4387	211	1366	2987	1343
1998	$3 803 526	5761	122	2244	4763	1023
1997	$3 433 839	4264	225	2499	4036	1020
1996	$3 309 192	4544	297	1367	2806	953
1995	$3 061 806	3725	156	1188	3465	496
1994	$2 996 299	4821	175	1940	3746	778
1993	$4 578 521	5120	160	1868	4192	635
1992	$3 084 931	3822	159	1427	3056	390
Total	$58 572 164	65 439	2659	24 288	51 541	13 212

*SAR costs are not adjusted for inflation.

Fig. 1. Statistics on search and rescue between 1992 and 2007. Table taken from "Dead men walking: search and rescue in US National Parks" [1].

With the development of search-and-rescue operations, satellite technology, and safety training, the United States' responses to disasters have improved in recent years. There is still, however, room for improvement. According to the Federal Emergency Management Agency, approximately 60% of American adults have not prepared for disasters through emergency drills, and merely 39% have created emergency plans [2]. This is troublesome because of the high costs associated with search and rescue (SAR), a type of disaster response. In "Dead men walking: search and rescue in US National Parks" and "Search and rescue trends and the emergency medical service workload in Utah's National Parks," Travis Heggie and his colleagues reported that, between 1992 and 2007, the United States spent a total of $58,572,164 on SAR. Additionally, they reported that 2,659 people were lost and 24,288 were injured during search-and-rescue missions [1, 3]. Figure 1 shows their tabulated findings. Similarly, the Bureau of Transportation Statistics found that, between 1985 and 2013, United States Coast Guard search and rescue teams–despite saving an average of 5,879 lives per year–were not able to save an average of 842 lives per year [4].

Robots can be impactful and have tremendous potential. In the past three decades alone, researchers have leveraged them to make deliveries in urban areas, to surveil large areas, and to conduct surgeries [5–7]. It follows that robots could improve disaster response–more specifically, search and rescue. Sonia Waharte and Niki Trigoni discovered that unmanned aerial vehicles (UAVs) are able to effectively map terrain and support search-and-rescue missions, especially if their search algorithms use partially observable Markov decision processes [8]. However, indoor environments are not often accessible to UAVs and present different navigation challenges than the ones that are. For this reason, we investigated the problem of programming robots to autonomously explore and map unknown indoor environments while localizing themselves within those environments. This problem may arise in situations where search and rescue operators seek to develop awareness of a collapsed and/or burning building without risking their lives to explore and map it. In the context of our investigation, the problem was most related to the CS 286 topics of path planning and simultaneous localization and mapping (SLAM),

as well as the topic of frontier detection.

II. Literature Review

The problem of programming robots to autonomously explore and map novel indoor environments is increasingly prevalent and important in robotics literature. There are two popular approaches to this problem. Some researchers explore the topics that relate to the problem–frontier detection and path planning–individually. However, some explore them together while working to achieve and extend SLAM. Below, we review literature relevant to both approaches–expounding works on frontier detection and works on path planning, then works relating to SLAM.

A. Frontier Detection

A robot that needs to autonomously explore and map a novel indoor environment needs to do so efficiently; it is vital for the robot to know which area to explore, and for that area to be valuable. An area may be valuable if it is: accessible to the robot, enlightening to the robot (meaning that it gives the robot a lot of new information about the map), or both. The valuable area that should be explored by the robot–and that necessarily borders explored and unexplored space–is defined as a frontier. Naturally, algorithms that return frontiers, given a map and a robot's position within it, are called frontier detection algorithms. A greedy frontier detection algorithm involves a breadth-first search of a robot's entire environment and returns the closest point of the robot's closest frontier. Needless to say, this is inefficient, as the closest frontier may not be best for long-term exploration of a map. While researchers have attempted to resolve this using edge detection and region extraction techniques from computer vision, these techniques require processing of the entire map at each time step, which is also quite inefficient. As a result, in "Robot Exploration with Fast Frontier Detection: Theory and Experiments," Matan Keidar and Gal Kaminka present a novel algorithm for frontier detection: the Wavefront Frontier Detection (WFD) algorithm [9]. The WFD algorithm is a graph search algorithm that involves two nested breadth-first searches; the first detects all frontiers within the robot's explored space and the second detects, among those frontiers, the one that would allow the robot to explore the greatest area. The WFD algorithm ultimately returns the point that is nearest to the centroid of its detected frontier. Importantly, Keidar and Kaminka noted that the WFD algorithm should be slow on large maps, because it requires a search over all space, but did not state exactly how slow it should be.

B. Path Planning

After getting a frontier, the robot needs to determine how to best navigate to that frontier and thereby faces a path planning problem. Like many other researchers, Huijuan Wang, Yuan Yu, and Quanbo Yuan used Dijkstra's algorithm–a greedy, graph search algorithm that returns the shortest path between two points–to solve this problem [10]. However, because Dijkstra's algorithm searches all possible paths, researchers' general preference has recently shifted toward the A* algorithm, which instead uses a heuristic function to prioritize points and paths. As Daniel Foead and his colleagues state in "A Systematic Literature Review of A* Pathfinding," the A* algorithm is now widely used for path planning problems due to its efficiency, simplicity, and modularity [11].

C. (Single-Robot) SLAM

Essentially, the SLAM problem is as follows: Is it possible for the robot to map an unknown environment while simultaneously inferring its location within that environment? Answering this problem is vital to the creation of truly autonomous robots, so it has become extremely popular in recent literature. In "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms", Hugh Durrant-Whyte and Tim Bailey provide a framework for single-robot SLAM, and present ways to handle landmarks, control inputs, and probability distributions [12]. They also explain algorithms, such as EKF SLAM. These algorithms are now incredibly popular. However, they crucially do not involve frontier detection or path planning. They instead go hand-in-hand with frontier detection and planning when researchers are attempting to create autonomous systems.

III. Baseline

For this project, we focused on three key topics: frontier detection, path planning, and SLAM.

We focused on frontier detection in simulation and with hardware. In doing so, we built on the greedy algorithm and the WFD algorithm. In simulation, we extended the algorithms such that they could be used with multiple robots. We also integrated them into our own infrastructure, which included custom environments and graphics. With hardware, we extended the algorithms such that they could overcome ROS-related issues, such as noisy data and errors. As mentioned above, the baseline greedy algorithm involves a breadth-first search of a robot's entire environment and returns the closest point of the robot's closest frontier. Meanwhile, the baseline WFD algorithm involves two nested breadth-first searches; the first detects all frontiers within the robot's explored space and the second detects, among those frontiers, the one that would allow the robot to explore the greatest area. The baseline WFD algorithm ultimately returns the point that is nearest to the centroid of its detected frontier.

Meanwhile, we focused on path planning in simulation, since the ROS navigation stack handles it with hardware. In doing so, we implemented the A* algorithm. This algorithm is typically formulated in graph-theoretic terms. It involves the extension of multiple trees (paths) from a start node (point) until one of them reaches an end node (point). In this extension, the algorithm minimizes a cost function that is related to the path length.

Additionally, we focused on single-robot SLAM with hardware. To do so, we integrated the aforementioned frontier detection algorithms with the Robot Operating System (ROS) navigation stack and SLAM, move_base, costmap_2d, and

rtabmap packages. (We also integrated the algorithms with ROS' multirobot_map_merge package, but multi-robot SLAM was severely limited by noisy measurements and unnecessary in our relatively small test bed. So, we eventually decided to concentrate on single-robot SLAM.) Doing this allowed a robot to autonomously detect frontiers and navigate to them, all while mapping its surroundings and localizing itself.

IV. Simulation Study

A. Overview

The primary goal of our simulation study was to compare the greedy frontier detection algorithm and WFD algorithm, focusing specifically on time until completion, in different controlled environments and with different numbers of robots. For this reason, we developed our own simulation infrastructure that relied on a two-dimensional map and incorporated all relevant mechanics, including robot sensing. At every time step, each robot senses its environment, runs its frontier detection algorithm, and makes a control decision based on the path to its detected frontier. Figure 2 shows a snapshot of our simulation in action.

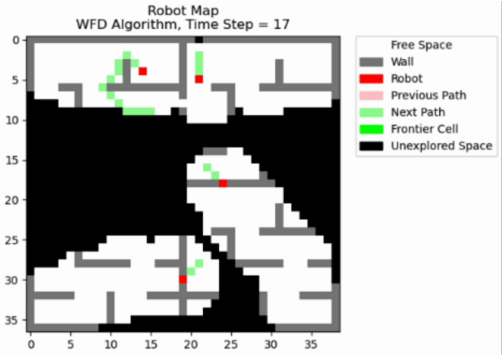

Fig. 2. A snapshot of the WFD algorithm in our simulation.

Below, we further detail how sensing, frontier detection, and path planning work in our simulation:

1) Sensing: We implemented sensing capabilities on the simulation by having a fixed map that is gradually revealed to the robot as it moves. At each time step, the robot executes a Scan() method, which reveals new information about the environment. To implement this, we used as input a text file of numbers that represented a two-dimensional grid occupied by three distinct types of objects: robots, obstacles, and free space. Our simulation takes this text file and interprets it as the grid class and instantiates Robot objects at the position they appear at in the grid. In the main loop of the mapping function, the robots begin by calling Scan. In the Scan functions, the robots use the ray casting heuristic to record a view of their environment. We use ray casting to implement realistic LiDAR sensing.

For the first part of ray casting, we need to determine how many angles are relevant to the robot. For the case of a two dimensional grid, the number of relevant angles, n, is determined by the perimeter of the sensing square around the robot. This can be calculated by the following equation:

$$n = 4(2r + 1) - 4$$

where r is the "sensing-radius" of the robot. From there, we can generate n equidistant "relevant" angles between 0 and 2π. Then, for each angle θ in the relevant angles, a point on the line formed by θ and the robot's position that is a distance r away from the robot's position is saved. Each "line" represents a LiDAR ray.

Next, for each point p in the saved points, Bresenham's creates a list of grid locations from robot's position to p. While iterating through the locations, each grid location is added to the set of known grid locations K if it is a free space. If it is not a free space, it is added to K, but the iteration terminates.

To clarify this entire process, the robot is able to scan by casting rays out in all directions, rays which terminate when they reach a distance of r of collide with an obstacle. All grid locations that these rays cross are now known to the multi-robot system.

2) Frontier Detection: After completing a scan, the robot then finds a frontier. Recall that a frontier is a known position that borders an unknown position. We have attempted frontier detection with two different algorithms. The naive approach finds a frontier using a single Breadth First Search (BFS) implementation. Starting with the robot's position, we exhaustively search all neighboring points until we reach a point that is not in K, indicating that it neighbors a frontier. Furthermore, by the properties of BFS, we know that this is the closest frontier to the robot.

The second approach is with the Wavefront Frontier Detection algorithm (WFD) from "Robot Exploration with Fast Frontier Detection: Theory and Experiment". This algorithm uses a nested BFS approach to find the frontier with the largest unknown area.

3) Path Planning and Control: Lastly, with this frontier defined, we use the A* algorithm to find the shortest path to the frontier without colliding with obstacles.

B. Assumptions

Here, we make a few assumptions about the environment and the robots. In a particular experiment, we assume that the environment is unknown but static. That is, the environment does not change as the robot moves. Furthermore, we only use building-like environments, in an effort to replicate urban search-and-rescue operations. Additionally, we assume that the robots can acquire perfect information about the environment in a fixed "sensing-radius" around them, and can flawlessly share this information with other robots. Finally, we assume that the robots know their initial positions and can accurately update their positions at each time step.

C. Results

As shown in Figure 3, the WFD algorithm was slower than the greedy algorithm, especially as the size of the map area increased. This was expected. As described above, according

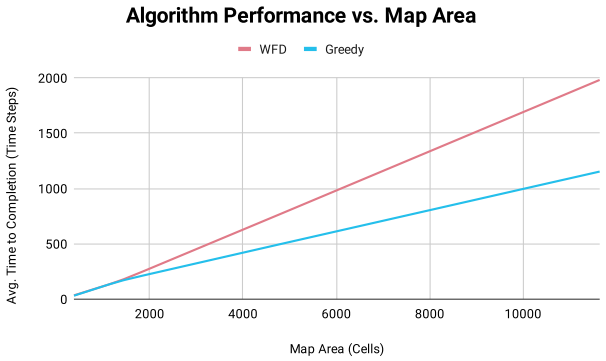

Fig. 3. A graph displaying how the WFD and greedy algorithms' overall simulation performance, in terms of time to completion, relates to map area.

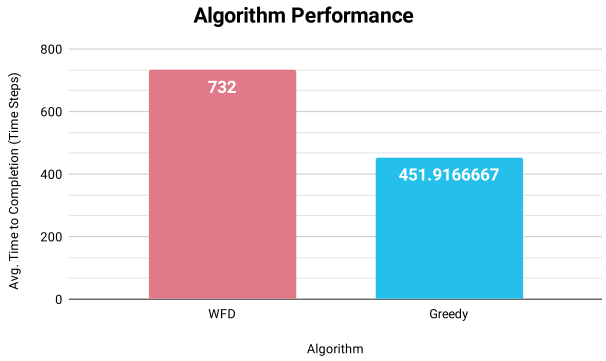

Fig. 5. Overall simulation performance, in terms of time to completion, of the WFD and greedy algorithms. Values were averaged across three different types/sizes of maps and four different team sizes.

to "Robot Exploration with Fast Frontier Detection: Theory and Experiment," it is likely because the WFD algorithm searches through all explored space for the frontier with the largest area [9]. Meanwhile, the greedy algorithm can terminate as soon as it detects a single frontier.

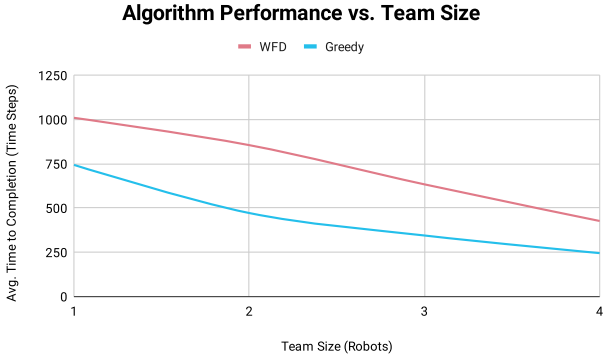

Fig. 4. A graph displaying how the WFD and greedy algorithms' overall simulation performance, in terms of time to completion, relates to team size (that is, the number of robots working together within a map).

Figure 4 and Figure 5 show a similar trend: The WFD algorithm was outperformed by the greedy algorithm with different team sizes (that is, with different numbers of robots working together in a map) and in general. Now, this is very likely due to the WFD algorithm's poor performance on large maps, which is positively skewing its average time to completion across all cases. However, it could also be a consequence of the maps that we chose to use. To reiterate, we chose to use building-like maps, which tended to cause robots to oscillate between small rooms when using the WFD algorithm. This suggests that the WFD algorithm may be better suited for relatively open maps.

Nevertheless, we see an overall trend with both algorithms in terms of time to completion and team size. The average time to completion decreases as team size increases, but the efficiency gain from adding a new robot decreases as the number of robots increases. This can be explained by the clustering of the robots, as more of the map gets explored. After a large portion of the map gets explored quickly, the robots tend to explore frontiers together. Thus, adding more robots has diminishing returns. This issue could changed using task-allocation–assigning different frontiers to different robots whenever possible.

V. Hardware Experiments

A. Overview

In our hardware experiments, we built on the aforementioned frontier detection algorithms and integrated them with ROS on a single robot. (We initially implemented the algorithms on multiple robots, but had to change course after realizing that multiple robots were not required to map our relatively small test bed and produced extremely noisy maps.)

In our implementation of the frontier detection algorithms, we rely on the robot's occupancy grid, cost map, and pose. After all three of these are published and synchronized, we convert them from a real coordinate system to a grid coordinate system. We also binarize the cost map. We then combine the occupancy grid and cost map to create a merged map that includes both unexplored areas and dilated obstacles. We do this to ensure that any point returned by a frontier detection algorithm is theoretically reachable. After adding the robot's position to this merged map, we feed the map into either the WFD or the greedy frontier detection algorithm.

As mentioned above, the baseline WFD algorithm returns the nearest-to-centroid point of the frontier that would allow the robot to explore the greatest area. However, because the ROS navigation stack often misclassifies the reachability points, we built on the WFD algorithm to make it suitable for error handling. In particular, we modified the algorithm such that it returns a list of candidate frontiers. We sort this list in terms of the frontiers' sizes, from biggest to smallest. Next, we reduce this list to only include points that are equal to, or nearest to, the centroids of the frontiers. In this way, the resulting list essentially includes candidate frontier cells–points belonging to frontiers–that are sorted from

greatest to least "impact." We iterate through this list, checking if the current candidate frontier cell is reachable (i.e., if the ROS navigation stack can successfully plan a complete and valid path to it). If it is, we send the frontier cell as a goal to the robot. If it is not, we instead send the earliest reachable point on the path to the frontier cell. If no such point exists due to noise in the ROS navigation stack, we move on to the next candidate frontier cell and repeat this procedure.

We built on the greedy algorithm in a nearly identical way, but preserved its differences from the WFD algorithm. Our revamped version returns a list of the robot's closest frontier cells that belong to sufficiently large unexplored areas and that are sufficiently distant from one another. To optimize the algorithm, we only allow this list to contain up to 20 frontier cells. We handle this list just as we handle the list that comes from the WFD algorithm (the one containing candidate frontier cells that are sorted from greatest to least "impact").

If the frontier detection algorithm that we employ either returns no frontiers or if it returns frontiers that can not be reached due to noise in the ROS navigation stack, we command the robot to randomly explore the map (safe wander), navigating to reachable points within a certain radius. Each time it safe wanders, it re-runs its frontier detection algorithm. If the robot still lacks reachable frontiers and intermediate points after safe wandering three times or if the robot cannot safe wander at all, it has almost certainly completed its map, so we command it to idle.

Through the algorithms, we effectively equipped the robot to autonomously explore and map novel indoor environments and, in doing so, solved the problem described in Section I. A robot with this capability could undoubtedly support search-and-rescue operators, perhaps giving them knowledge (i.e., a map) of an indoor environment before they enter it.

Like we did in our simulation, we studied and compared the overall performance, in terms of time to completion, of the greedy and WFD algorithms. We first had one robot use the greedy algorithm to autonomously explore and map a fixed environment, displayed in Figure 6, five times. We then computed the average time taken by the robot to completely do this, replaced the greedy algorithm with the WFD algorithm, and repeated the procedure.

Fig. 6. An overhead map of the environment in which we ran our hardware experiments.

B. Assumptions

Throughout the experiments, both algorithms dealt with some noisy data and errors in the ROS navigation stack, but roughly to the same degree. We assumed that, throughout five trials, the average effect of the noisy data and ROS errors on the greedy algorithm would be equal to that on the WFD algorithm. Thus, we did not account for it in our calculations (though we do acknowledge it in our analysis). Notably, we make no other assumptions except, of course, the assumption that our robot is compliant with our algorithms.

C. Results

Both algorithms resulted in effective and autonomous frontier exploration and mapping. The greedy algorithm's only trouble came from noisy data and errors in the ROS navigation stack, as shown in this video, which showcases one of our five trials. The same was true of the WFD algorithm, as shown in this video, which showcases another one of our five trials.

At the end of our five trials, we discovered that the WFD algorithm exhibits better average performance, in terms of time to completion, than the greedy algorithm. In fact, as shown in Figure 7, it resulted in 18.2% faster exploration and mapping, on average.

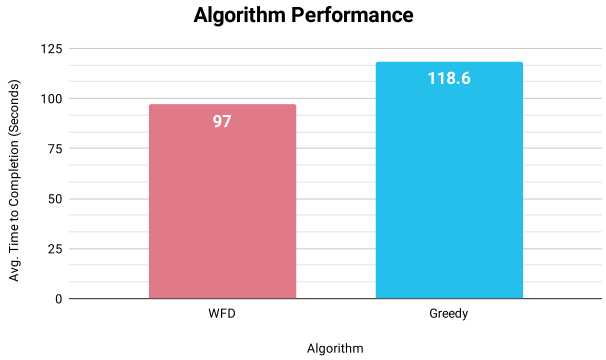

Fig. 7. Overall real-world performance, in terms of time to completion, of the WFD and greedy algorithms. Values were averaged across five trials on one map.

These results aligned with Keidar and Kaminka's introduction of the WFD algorithm; it indeed performs better than the greedy algorithm on small maps. However, we must acknowledge that the results may be partially explained by noisy data and/or errors in the ROS navigation stack. The greedy algorithm could have detected small frontiers that did not actually exist and that led to inefficient exploration. All the while, the WFD algorithm detected the largest frontiers that led to more efficient exploration.

These results were especially interesting because they contradicted some results of our simulation study. This indicates that the greedy and WFD algorithms' performances depend heavily on the assumptions made in measuring them. It also implies a need for further application-based research on the frontier detection algorithms; we imagine that these results

may vary with different real-world map sizes and configurations.

All in all, this demonstrates that relatively inexpensive robots like LoCoBots are capable of autonomously exploring and mapping novel indoor environments.

VI. Wildcards

Because, throughout this project, we focused heavily on our extended version of the WFD algorithm, we chose to use it for three wildcards. More specifically, we chose to deploy the algorithm in a dark environment, in a new environment, and in both a dark and new environment. These three wildcards challenged the algorithm by forcing it to only rely on LiDAR data and to completely lack assumptions about its input. They also aligned perfectly with the problem that we hope to solve; one can imagine that the indoor environments explored and mapped by robots in search-and-rescue scenarios are very rarely known or well-lit.

The WFD algorithm ultimately succeeded when faced when all three wildcards. Its success is highlighted by this video, which shows the algorithm in a dark environment; by this video, which shows the algorithm in a new environment; and in this video, which shows the algorithm in both a dark and new environment.

However, this is not to say that the WFD algorithm never failed. Extending the algorithm for these wildcards was extremely difficult. We often found that, after the robot received a perfectly valid waypoint, according to the WFD algorithm (or the greedy algorithm, for that matter), it would spin in place and "refuse" to move. The robot would do so because it incorrectly believed that the waypoint was not reachable because of noise in the robot's sensor data, because of error in the ROS navigation stack, or because the robot's path planning and map creation are done somewhat independently. This was especially common on crowded maps, such as the one displayed in the above videos. We were able to overcome this challenge by using the ROS navigation stack's path plan to robustly check whether a waypoint is reachable.

While building on the WFD algorithm for these wildcards was extremely difficult, it was also incredibly enlightening. Because we decided on the wildcards early, we integrated all of our ideas with the ROS navigation stack, as well as the ROS SLAM, move_base, costmap_2d, and rtabmap packages. Our hands-on, trial-by-fire integration was unlike anything we had ever done, and demanded that we not only apply, but also expand, our knowledge of ROS.

exploring and mapping novel indoor environments in search-

VII. Conclusion

In summary, we investigated the problem of programming robots to autonomously explore and map novel indoor environments while localizing themselves within those environments. In particular, we built on frontier detection in simulation and with hardware; implemented path planning in simulation; and achieved SLAM with hardware. Ultimately, we showed that relatively inexpensive robots are capable of autonomously and-rescue scenarios. Additionally, in simulation, we studied and compared the WFD algorithm and a greedy frontier detection algorithm–characterizing the extent to which the WFD algorithm is slower on large maps. This had not previously been done. Furthermore, with hardware, we studied and compared the WFD and greedy algorithms–determining that the WFD algorithm is actually faster in small, real-world environments.

In the future, this project could be furthered in several ways. Firstly, one could combine our work with task allocation to avoid robot clustering. Secondly, one could test our work with additional types of maps, additional sizes of maps, and additional team team sizes. Thirdly, one could apply our work to multi-robot SLAM with hardware. Fourthly, one could write a proprietary ROS navigation stack which does not have any of the current issues.

Acknowledgements

We are grateful for the unwavering support and guidance of the entire CS 286 teaching staff. The opportunity to investigate the ways in which robotics and search and rescue could be bridged was remarkably fulfilling and would have been impossible without them.

References

[1] T. W. Heggie and M. E. Amundson, "Dead men walking: search and rescue in us national parks," *Wilderness & environmental medicine*, vol. 20, no. 3, pp. 244–249, 2009.

[2] G. Darch, "Ready arlington: Why don't people prepare themselves for emergencies?" *ARLnow*, 2016. [Online]. Available: https://www.arlnow.com/2016/09/01/ready-arlington-why-dont-people-prepare-themselves-for-emergencies/

[3] T. W. Heggie and T. M. Heggie, "Search and rescue trends and the emergency medical service workload in utah's national parks," *Wilderness & environmental medicine*, vol. 19, no. 3, pp. 164–171, 2008.

[4] "U.s. coast guard search and rescue statistics, fiscal year," 2014. [Online]. Available: https://www.bts.gov/archive/publications/national_transportation_statistics/table_02_49

[5] A. Buchegger, K. Lassnig, S. Loigge, C. Mühlbacher, and G. Steinbauer, "An autonomous vehicle for parcel delivery in urban areas," in *2018 21st international conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2961–2967.

[6] E. E. Joh, "Private security robots, artificial intelligence, and deadly force," *UCDL Rev.*, vol. 51, p. 569, 2017.

[7] C. N. Gutt, T. Oniu, A. Mehrabi, A. Kashfi, P. Schemmer, and M. W. Büchler, "Robot-assisted abdominal surgery," *Journal of British Surgery*, vol. 91, no. 11, pp. 1390–1397, 2004.

[8] S. Waharte and N. Trigoni, "Supporting search and rescue operations with uavs," in *2010 International Conference on Emerging Security Technologies*, 2010, pp. 142–147.

[9] M. Keidar and G. Kaminka, "Robot exploration with fast frontier detection: Theory and experiments," vol. 1, 06 2012, pp. 113–120.

[10] H. Wang, Y. Yu, and Q. Yuan, "Application of dijkstra algorithm in robot path-planning," in *2011 second international conference on mechanic automation and control engineering*. IEEE, 2011, pp. 1067–1069.

[11] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, and E. Gunawan, "A systematic literature review of a* pathfinding," *Procedia Computer Science*, vol. 179, pp. 507–514, 2021, 5th International Conference on Computer Science and Computational Intelligence 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921000399

[12] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

Group 6 Final Project Instructions

CS 286

Spring 2022

Simulation

To run our simulation code/to replicate our simulation results, please do the following:

- Change the working directory to “Simulation”
- Run “python3 simulation.py”
    - Install dependencies if necessary

Note: This may take up to 90 minutes!

Hardware

To run our hardware code/to replicate our hardware experiments, please do the following:

- Run roscore
- Connect to a LoCoBot after turning it on and placing it in the test bed
- Launch RVIZ
- Add a pose to RVIZ, which shows the robot’s current_goal
- Change the working directory to “Hardware”
- Open main.py
- If you would like, FD_ALGO and/or ROBOT_NAME using the guide on line 25
- Run python main.py