Autonomous Reconnaissance for Disaster Relief

Ahmet Demirkaya
Electrical and Computer Engineering
Northeastern University

Hardik Devrangadi
Electrical and Computer Engineering
Northeastern University

Beyza Kalkanli
Electrical and Computer Engineering
Northeastern University

Kyle Lockwood
Electrical and Computer Engineering
Northeastern University

Krishna Prasath Senthil Kumaran
Electrical and Computer Engineering
Northeastern University

***Abstract*—Disaster relief efforts can be dangerous for human rescuers. Rescuers may need to perform exhaustive searches of disaster areas when searching for victims, exposing themselves to life-threatening hazards. Disaster areas encompassing large areas will require prohibitively long search times, which can be fatal when victims depend on immediate rescue. This makes disaster reconnaissance an ideal candidate for automatization. Autonomous reconnaissance requires a robotic agent to optimally search the disaster area and identify victims. This identification provides an entry point of rescue for human intervention. In our setup, we simulate a disaster environment using a Turtlebot as an autonomous rescue agent, and AprilTags to represent victims. We propose a customized searching and mapping algorithm that attempts to minimize rescue times and prevents search termination before all victims are identified. In this approach, we repeatedly map the disaster environment using the explore_lite ROS package. After each complete mapping, explore_lite is reinitialized with the robot starting in the least-explored region of the map. This reinitialization is determined using a heatmap generated from the path the robot follows during the previous mapping iteration. We simulate this disaster rescue scenario problem in our setup and assess how variations of our approach detect and localize AprilTags.**

***Index Terms*—**

I. Introduction

Disaster reconnaissance represents an ideal candidate for automatization due to lengthy search times and high risks incurred by human rescuers. Though relatively simple in theory, this problem requires joining mapping and searching approaches while balancing high consequences for failure. Existing approaches are capable of searching and mapping environments with obstacles and occluded targets, however, these approaches do not always address the requirements unique to this problem. Critically, any search for disaster victims cannot terminate until all victims have been discovered and the approach to mapping must accommodate a potentially dynamic environment.

There have been several recent attempts to apply the mapping and searching problem to recconnaissance of disaster environments using networks of robots. Seenu et al. utilize Particle Swarm Optimization to allow a network of mobile robotics to search areas affected by earthquakes and tsunamis [1]. Dogru et al. explore autonomous multi-robot systems applied to Search and Rescue tasks in unstructured complex environments [2]. Inspired by the Fukushima tsunami and nuclear disaster, Kuntze et al. developed a sensor network supported by mobile robots to accelerate victim search and rescue [3]. Other groups have attempted to automatize damage assessment to inform future recconaissance efforts. Ghosh et al. use convolutional neural networks to visually assess types of damage to buildings following an earthquake [4]. This type of information could be adapted to our approach when assigning priorities for future areas to explore. One variation of our approach prioritizes areas surrounding obstacles, based on the assumption that victims can be trapped by falling debris during a disaster event. Damage analysis could help inform this approach, as well as determine whether particular structures are in danger of imminent collapse.

One critical element in the design of search and rescue systems is the role that humans play in the operation and deployment of the robots. Chiou et al. carried out a robot-assisted nuclear disaster response exercise involving a human operator. They investigated how robotic platforms can be made more robust to reduce performance degredation resulting from cognitive and attentional demands on the operator [5]. The extent to which humans are able to participate in a robot-assisted rescue effort varies widely across different types of disasters. In our simulated disaster environment, we attempted to automatize the redeployment of the turtlebot after the completion of each mapping iteration. However, we found that this redeployment could more easily be carried out by a human. In certain disaster environments where robots can be deployed from different locations surrounding a search ”zone”, having a human in the loop may expedite the search process. This can be particularly useful when multiple robots are available, since new robots could be deployed immediately rather than waiting for the same robot to map the environment, plan a path to the redeployment location, then navigate to that location. A fully automatized approach may however be necessary in disaster environments (such as irradiated zones following nuclear disasters) where the risk to human rescuers is too high for them to enter or even approach the disaster area.

Evaluating the performance of rescue robots is a difficult task due to the variability across robotic platforms as well as the types of environments and disasters they are applied

to [6]. Murphy et al. note that simple quantitative metrics often fail to capture the value of a rescue robot's findings. For example, a metric such as "number of survivors found" penalizes a robot for determining that a particular area contains no survivors, even though this may be useful information to the rescue effort. They also point out that computer and physical simulations attempting to predict robot performance are difficult to validate since they are much easier to complete successfully than a real-world robot response. We selected three simple metrics to quantify the performance of our approach which take into account the specific goals of the problem. These metrics include total time to find all AprilTags, average time to find each individual AprilTag, and total path length traveled by the robot.

II. Environment Description

We have created virtual and real-world simulations of a disaster environment. For the virtual simulation, we utilized the Gazebo toolbox where we can create an environment, import the Turtlebot, and generate multiple scenarios for our experiments . For those simulations, six victims are represented by AprilTags in both simulations. In the virtual simulation, the AprilTags are distributed to various locations and in the real-world simulation, they are pasted to various surfaces. Nine obstacles are used in the virtual simulation whereas the real-world simulation has six obstacles that occlude the AprilTags when viewed from different angles. Fig. 1 and Fig. 2 are the virtual and real-world simulations respectively.

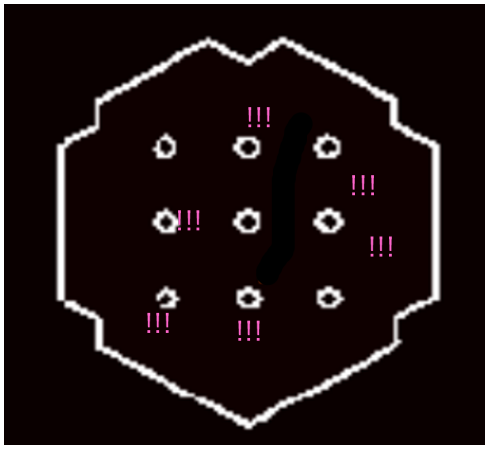

Fig. 1. **Virtual Disaster Environment setup**: The virtual environment setup consists of nine obstacles organized in a 3x3 grid pattern. Six AprilTags (indicated with purple exclamation marks) were scattered throughout the environment around these obstacles.

III. Proposed Approach

The core of our approach involves repeatedly mapping the disaster environment using the explore_lite package, and using a variety of reinitialization strategies to determine where the robot will be redeployed for each mapping.

Fig. 2. **Real World Disaster Environment setup**: Our real world environment setup contained six obstacles represented by trash bins and cabinets. We used six printed AprilTags to represent victims.

A. Version 1:

In the first version of our approach, we will allow explore_lite to complete a map of the environment while detecting AprilTags in parallel. Assuming we know the number of "victims", we will simply restart explore_lite following completion of mapping until all AprilTags are discovered. Each new iteration of explore_lite will begin with the robot initialized at whatever point in the environment it finished at from the previous iteration.

B. Prioritize Least-Explored Area:

AprilTags may be missed even if the area around them is explored. This can be due to their orientation or positioning next to an obstacle. The second version of our approach addresses this by changing our redeployment strategy after each run of explore_lite to prioritize areas the robot visited the least. We generate heatmaps of the environment after each run of explore_lite based on the path the robot takes. Areas the robot has visited multiple times receive a higher weight in the heat map. After explore_lite finishes, the robot is directed to the "least-visited" area to start for the next iteration. AprilTags are easier to detect if we approach them from multiple angles. If the robot spends a longer time in one part of the environment, we can assume that it had the opportunity to approach any AprilTags in that area from these multiple angles. For areas of the environment that the robot explored but spent less time, it is more likely that these tags were missed due to inadequate exploration of those areas. This can also address noisy estimations of AprilTag pose. Using the heatmap from the robot path we can establish a confidence in the pose estimation of each tag. If a tag has been discovered but is in a relatively unexplored area (maybe it was only viewed from one angle, etc), we can say we have less confidence in the pose. We can leverage pose confidence of AprilTags as a second condition for restarting explore_lite

(i.e. explore_lite will repeat until we have met this threshold for all AprilTags). In our current implementation, we repeat exploration until all AprilTags are discovered, but do not enforce a confidence threshold on AprilTag pose. The resulting heatmaps and redeployment locations using this method are shown in Figure 3.

C. Reinitialization

When generating the heatmap, the robot path is additive. If the robot explores an area multiple times, that area is given a higher weight because the chances of finding an AprilTag in that area are higher (approaching from different angles, etc). After each run of explore_lite we return an occupancy grid for the environment and the robot's path. We use the robot path to generate the heatmap. Each point in the occupancy grid is given an initialization priority based on its own proximity to the robot path from the previous iteration, as well as the proximity of its neighboring points in the grid to the robot path. Points that are farthest away from the path receive the highest initialization priority. The robot is initialized for the next iteration of explore_lite at the location with the highest initialization priority.

D. Prioritize Areas Around Found AprilTags

In a disaster environment, victims may be clustered together (i.e. multiple members of a family). In this situation, we may want to give a greater initialization weight to the areas surrounding discovered AprilTags (the regions surrounding AprilTags are treated as unexplored). The results of these simulations can be seen in Figure 3.

E. Prioritize Areas Around Obstacles

In certain disaster scenarios, it may be likely to find victims hidden or trapped by obstacles. We can treat the areas surrounding discovered obstacles as unexplored when reinitialization the robot after each run of explore_lite.

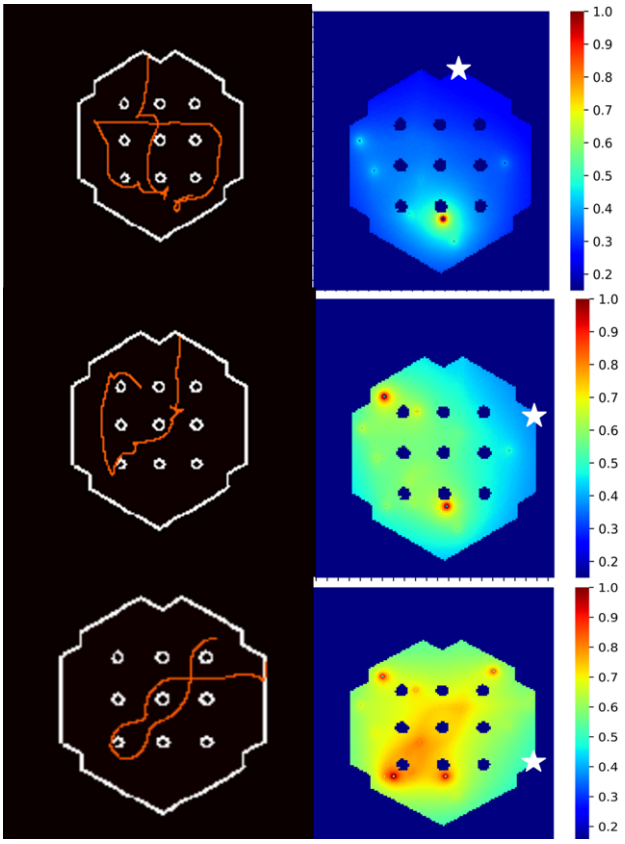

Fig. 3. **Simulation results when prioritizing least-explored areas**: Each row represents a separate run of explore_lite (3 runs in total). The left figure in each row shows the occupancy grid with the robot path overlaid. The right figure shows the heatmap generated from the robot path. The selected redployment location for the subsequent iteration is indicated with a white star. For example, row 2 uses the redeployment location shown in row 1, and row 3 uses the location from row 2. In this redeployment strategy we generate a heatmap covering the entire area with intensities based on proximity to the robot's path. This heatmap is cumulative, so areas that the robot spends more time in will be weighted less when choosing a new location for redeployment.

IV. Discussion and Results

: We tested our three different redeployment strategies in simulation: prioritizing areas least-visited by the robot, prioritizing areas around found AprilTags, and prioritizing areas around obstacles. Interestingly, we found that each method resulted in distinct reinitialization patterns.

When prioritizing least-visited areas, we found that the model frequently selected areas on the outer edge of the environment for redeployment. This pattern makes sense given that the robot almost always travels through the center of the environment during mapping, and these edge-regions are often the farthest away from the most concentrated zones of robot exploration. Figure 3 shows the generated heatmaps for three mapping iterations utilizing this redeployment strategy. By the third iteration, we can see that the majority of the environment has been marked as explored.

When prioritizing areas around found AprilTags, we observed that the redeployment point was almost always directly next to one of those AprilTags. This behavior is expected given our assignment of priorities. We did find, however, that using too large of an influence radius surrounding found AprilTags can lead to large parts of the environment being marked as unexplored. This may be acceptable when the environment is very large, however, in our smaller simulated environment we needed to decrease the influence of the AprilTags. Even then, we saw that the environment was almost entirely marked as unexplored by the time we discovered five or more AprilTags. One important note is that we did not purposely cluster our AprilTags together. To properly test this method, we would need to create separate simulated environments with different d

When prioriti

A common way AprilTags can be missed is if they are viewed from an incorrect angle and cannot be detected (i.e. side-on). This can occur even when their surrounding area is explored by the robot, which motivated the approaches we discussed previously. An additional measure to improve likelihood of detecting the tags is to have the robot pause

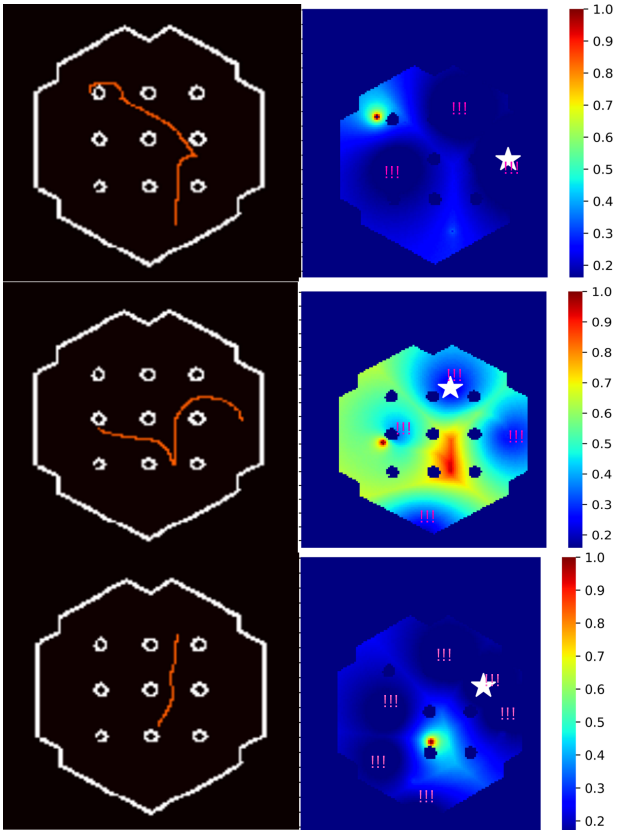

Fig. 4. **Simulation results when prioritizing regions around found April-Tags**: In this redeployment strategy, the areas surrounding found AprilTags are treated as unexplored. We can see that the heatmap shows more blue (unexplored) areas as additional tags are discovered. AprilTags are indicated using purple excalamation marks.

Tag ID	1	2	3	4	5	6
Round 1	4.73	-	4.73	4.73	-	-
Round 2	11.91	-	42.75	-	-	-
Round 3	-	-	19.29	-	-	-
Round 4	84.98	-	66.14	18.38	-	-

and spin around 360 degrees every 10 seconds. This can also improve our pose estimation of discovered AprilTags, as we will be more likely to view them from multiple angles.

V. Results

A. Simulation

1) Creation of Simulated Environment: Robot can detect AprilTags, collide with walls other obstacles, and be controlled by teleop in Rviz.

2) Generation of AprilTags in Gazebo (Single and Multiple AprilTags):

3) Detection of AprilTags in Rviz (Single and Multiple AprilTags): Simulation results when prioritizing least-explored areas

Simulation results when prioritizing regions around found April-Tags

B. Real world

1) Installation of Camera for AprilTag Detection:

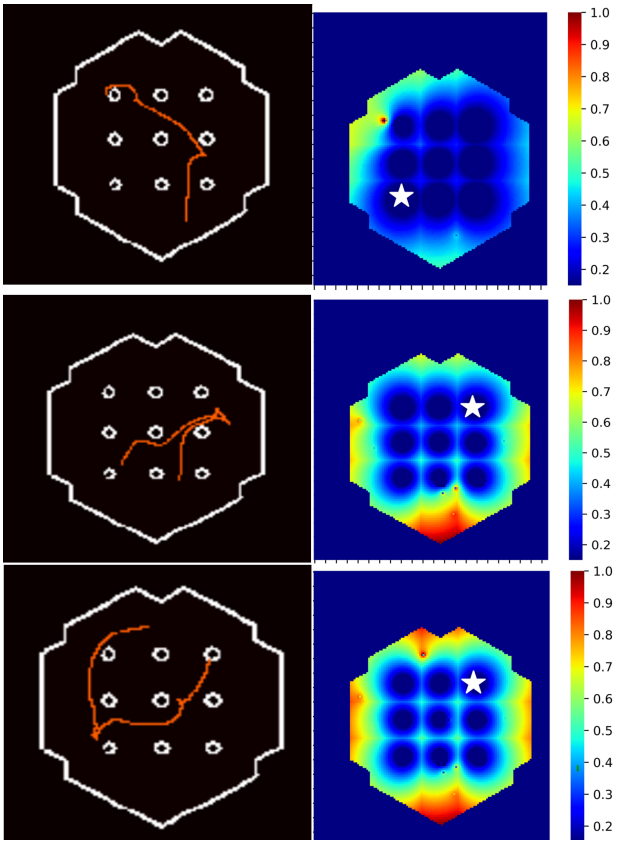

Fig. 5. **Simulation results when prioritizing areas around obstacles**:

Tag ID	1	2	3	4	5	6
Round 1	4.73	-	4.73	4.73	-	-
Round 2	-	-	19.29	-	-	-
Round 3	26.27	-	23.38	-	-	-

2) GMapping and Teleop in Rviz:

3) Using Teleop in Real World:

VI. Challenges Faced

During the implementation of the proposed approach, we encountered challenges at the following stages:

- Loading multiple AprilTags to the simulation environment
- RPi camera setup and calibration
- Identifying skewed or partial AprilTags
- Creation and implementation of a universal coordinate system, for heatmap processing and generating initialization points for successive iterations

VII. Strengths and Weaknesses

A. Strengths

Our termination condition ensures that the robot will continue remapping the environment until all AprilTags are found. If some tags are never found, the robot will continue searching. Prioritizing regions around AprilTags and obstacles can improve search times. Remapping the environment allows discovered AprilTags to be viewed from multiple angles,

reducing uncertainty in pose estimation. Disaster environments may be dynamic. Remapping the environment over multiple iterations reduces the risk of relying on an environment map that is no longer valid.

B. Weaknesses

Completely remapping the environment over multiple iterations is costly. Reinitializing in a particular area does not guarantee that the area will be adequately searched (although over enough repetitions, this should improve). If the true number of AprilTags is not known, a threshold must be set to tell the robot when to stop searching. This can be set arbitrarily high so that the robot never stops searching, which might be preferred in a large disaster area where we lack an accurate estimate of the number of victims.

REFERENCES

[1] N. Seenu, L. Manohar, N. M. Stephen, K. C. Ramanathan, and M. Ramya, "Autonomous cost-effective robotic exploration and mapping for disaster reconnaissance," in *2022 10th International Conference on Emerging Trends in Engineering and Technology-Signal and Information Processing (ICETET-SIP-22)*. IEEE, 2022, pp. 1–6.

[2] S. Dogru, S. Topal, A. M. Erkmen, and I. Erkmen, "A framework for prototyping of autonomous multi-robot systems for search, rescue, and reconnaissance," in *Prototyping of Robotic Systems: Applications of Design and Implementation*. IGI Global, 2012, pp. 407–437.

[3] H.-B. Kuntze, C. W. Frey, I. Tchouchenkov, B. Staehle, E. Rome, K. Pfeiffer, A. Wenzel, and J. Wöllenstein, "Seneka-sensor network with mobile robots for disaster management," in *2012 IEEE Conference on Technologies for Homeland Security (HST)*. IEEE, 2012, pp. 406–410.

[4] T. Ghosh Mondal, M. R. Jahanshahi, R.-T. Wu, and Z. Y. Wu, "Deep learning-based multi-class damage detection for autonomous post-disaster reconnaissance," *Structural Control and Health Monitoring*, vol. 27, no. 4, p. e2507, 2020.

[5] M. Chiou, G.-T. Epsimos, G. Nikolaou, P. Pappas, G. Petousakis, S. Mühl, and R. Stolkin, "Robot-assisted nuclear disaster response: Report and insights from a field exercise," *arXiv preprint arXiv:2207.00648*, 2022.

[6] R. R. Murphy, S. Tadokoro, and A. Kleiner, "Disaster robotics," in *Springer handbook of robotics*. Springer, 2016, pp. 1577–1604.