# Autonomous Robot Reconnaissance in Simulated Disaster Environment

Christian Burwell
burwell.c@northeastern.edu

Shaun Gentilin
gentilin.s@northeastern.edu

Robert Sylvia
sylvia.r@northeastern.edu

Bibek Gupta
gupta.bi@northeastern.edu

Ruiqi Zhang
zhang.ruiqi@northeasten.edu

***Abstract*—Reducing risks in disaster response is an important problem which can be remedied by a mobile robot that performs autonomous reconnaissance. The proposed solution involves the design of a robot that can localize itself in an unknown environment, create a map, and find victims. This is tested in a closed environment utilizing a Turtlebot3 and representing victims with AprilTags. The Turtlebot is designed to scan data, detect AprilTags and estimate poses, while a remote laptop workstation handles navigation and planning, SLAM, and exploration algorithms. Gmapping and Slam Toolbox were the packages experimented with to generate an occupancy grid map, and the latter was chosen for better accuracy and functionality. For AprilTag detections, averaged pose estimates were used with the apriltag_ros package to tackle the problem of tags being estimated from too far away and the limited field of view of the camera. The design was tested in a Forsyth Hall lab with five AprilTags at different heights and positions. All tags were found, and a map was generated successfully. It is concluded that the proposed system performs well against a benchmark using only off-the-shelf components, and that it addresses the issue of disaster response with detail and accuracy.**

## I. Introduction and Motivation

Disaster responses are extremely dangerous, particularly entering terrains where no prior intelligence information may be available. Autonomous reconnaissance robots are a great solution for reducing danger by providing imagery and maps of potentially hazardous environments, such as collapsed buildings. To address these needs, a reconnaissance system has been developed that is capable of exploring a closed environment, while generating a map and locating victims.

This system is using TurtleBot3 Burger for this task. It is a compact and customizable mobile robot designed to be used with ROS. With its 360° LiDAR, it can perform SLAM and navigation to explore the environment. AprilTags are the chosen representation for victims due to the simplicity in detecting them within an image. A suitable indoor lab is chosen as a test environment, where the robot's sensors can reach their full potential. Different solutions were tested on the robot to design the optimal reconnaissance system.

## II. Proposed Solution

### A. System Construction

For ease of development, the ROS network is split between the Turtlebot and a remote laptop workstation. This allowed lightweight and time critical programs to be run directly on the Turtlebot while heavier navigation calculations could be run on a system with more processing power.

On the Turtlebot side, the hardware drivers for the wheels, IMU, LiDAR, and camera were run along with code to facilitate AprilTag detection and pose estimation. As each of these tasks was rather light-weight, the Turtlebot's processor (a Raspberry Pi 3b) did not act as a bottleneck.

On the laptop's end, more intensive tasks were run such as the navigation planner, SLAM, and frontier exploration algorithms. Move_base Ros node has been utilized [1] for motion planning, slam_toolbox [2] for SLAM, and a modified explore_lite [3] to designate goals for frontier exploration. These combined made up the navigation stack.

While move_base and the localization component of the SLAM algorithm utilize the occupancy grid generated by slam_toolbox, an intermediary node modifies that occupancy grid with a custom mask before passing it to explore_lite's frontier exploration algorithm. To generate this occupancy grid the system utilizes the Turtlebot's estimated pose in the world frame and a camera view model to mask out unseen areas from the original occupancy grid mask. The robot's pose is given by SLAM, and the camera view model is generated by constraining the LiDAR's viewing angle and max range. The constraints put on the LiDAR were determined empirically by testing different angles and distances of the AprilTag detection system. With this information, the mask is generated by taking the union of each incoming transformed camera model and the previous mask. This mask is then applied to the original occupancy grid such that unoccupied space in the base occupancy grid only remains unoccupied if it falls within the regions designated by the mask. Occupied space and unexplored space remain unchanged. This modified occupancy grid is then passed to the frontier exploration algorithm.

In order to determine the pose of all AprilTags in the environment, the system runs the apriltag_ros package [4] while the Turtlebot explores. The apriltag_ros package analyzes image data from the camera to detect AprilTags, estimating and publishing that tag's pose relative to the camera frame. A custom node then receives this pose information. This node determines if a pose estimation is valid based on distance and deviation from the camera's z-axis and calculates the average pose over all previous valid pose estimates for the same tag after transforming them to the world frame. The average pose for each tag is then sent to ROS' TF tree.

### B. Design Rationale

Initially, the gmapping SLAM package was used to generate the occupancy grid, but later it was switched over to Slam Toolbox to address some issues. The gmapping-generated occupancy grid would occasionally make explore-lite set inaccessible goals. This would cause the turtlebot to stop exploring despite explore_lite being configured to choose a new goal if no progress is made for a period of time.

Slam Toolbox does not have this issue and its more complex loop closure behavior makes the map more globally accurate. Slam Toolbox also provides more configuration and parameter options than gmapping. Some examples being the ability to change the details of the solver used for loop closure and the option to run in a "lifelong" mode designed to operate over very large spaces for an extended period of time. These options initially caused issues with the performance of the SLAM algorithm, such as "lifelong" mode eliminating too much information due to similar environmental features and generating warped occupancy grids, but after enough tweaking produced a superior result. Ultimately, it was found that the "online sync" mode of the Slam Toolbox performed the best for our purposes.

If the greedy frontier exploration algorithm from the explore_lite package directly utilized the occupancy grid generated based on the 360-degree LiDAR, its exploration of the environment would only ensure the entire map has been seen by the LiDAR. This does not guarantee the camera has seen the entire environment and thus risks potentially missing AprilTags. However, discarding information from the scan to match the camera would severely impact the performance of both localization and move_base's planning algorithm. Thus, it is proposed to utilize the aformentioned mask before passing the occupancy grid to the exploration algorithm. The mask ensures that the exploration algorithm only marks areas seen by the camera as explored while not detrimentally influencing other parts of the system that utilize the base occupancy grid. With this modification, the frontier exploration behavior provided by the explore_lite package was sufficient. While explore_lite is not optimized to maximize the amount of the map seen by the camera, the mask ensures that exploration will continue until the camera has seen the entire map and the Turtlebot tends to prefer moving towards a goal such that the camera is facing forward. Therefore, the Turtlebot should eventually see the entire map with the camera.

Additionally, move_base offered all the functionality necessary for a path planner. As long as target goals were correctly specified and an accurate occupancy grid is constructed, move_base would typically behave as intended.

The proposed system seeks to average the pose estimates for each tag instead of just sending the outputs of the apriltag_ros package, which addresses a couple major issues. The first issue is that upon the system detecting a specific tag in an image, the existing package completely overwrites the previous pose estimation with the new one found from the current image. As a result, each pose estimation of the base system only uses the information from a single image to evaluate a tag's pose. Furthermore, the AprilTag 2 algorithm used by the package becomes more inaccurate with increasing distance and increasing deviation from the camera's central axis [4], especially the latter. This compounds the previous issue due to the last detection of a tag having an increased likelihood of being when the tag leaves the camera's field of view. Therefore, it seems a better approach to average the pose estimates to obtain a more accurate evaluation of the pose that wasn't reliant on data from a single image. Furthermore, an additional design approach was adopted where the system discards any pose estimates that fell outside the range of optimal measurements that has been pre-defined.

## III. Results

### A. Testing Environment

Fig 1. View of the testing environment from the Turtlebot

A good design also requires an appropriate test to show whether the design is actually working with a great level of accuracy. A mechanical engineering lab was used to create a test environment for the Turtlebot. The size of the configuration space was around 40 ft by 20 ft and contained 3 big tables and two trash cans as obstacles. These obstacles were kept in the middle of the lab with a good amount of vacant spaces between them and other obstacles were kept along the walls of the lab. A total of six AprilTags with different ID numbers were used to simulate disaster victims. Five AprilTags were placed on the walls around the lab, and one was stuck on one of the obstacles. The AprilTags were placed at different heights ranging from 10 inches to 20 inches from bottom of wall. To make the test environment more realistic, one AprilTag was placed on the wall and the line of sight to the tag was obstructed by a trash can placed just 1 foot away from the tag, while another AprilTag was placed under one of the tables and stuck near the leg of the table.

### B. Occupancy Grid Map

To determine the effectiveness of this system, a benchmark analysis was performed between a system created from only off-the-shelf components and our modified system. The off-the-shelf system refers to the system which uses gmapping and explore_lite as configured by the Turtlebot3 SLAM package and the base apriltag_ros package. The modified system is using SLAM Toolbox instead of gmapping, a masked occupancy grid to feed into explore_lite for frontier exploration, and a custom node to average AprilTag detections.

The gmapping-generated occupancy grid map (shown in Fig 2) has marked some spaces behind walls as both unoccupied space and on a frontier which explore-lite would potentially set as a goal, even though it is impossible for move_base to navigate to that space. This often caused the Turtlebot to stop exploring despite explore-lite being configured to choose a new goal if no progress is made for 30 seconds. Also, the features in the map are not super clear and shadowed areas can be seen where the Turtlebot's camera has not viewed. On other hand, Slam Toolbox creates much more defined wall boundaries, and its more complex loop closure

behavior makes the map more accurate as shown in Fig 3. The features in the map look clearer with more detailed definitions of their boundaries. There were also few shadow spots which shows the modified system explored all the spaces present in the test environment. The few spots in shadow would have been explored with more time, as shown by the Turtlebot navigating to one in Fig 3.

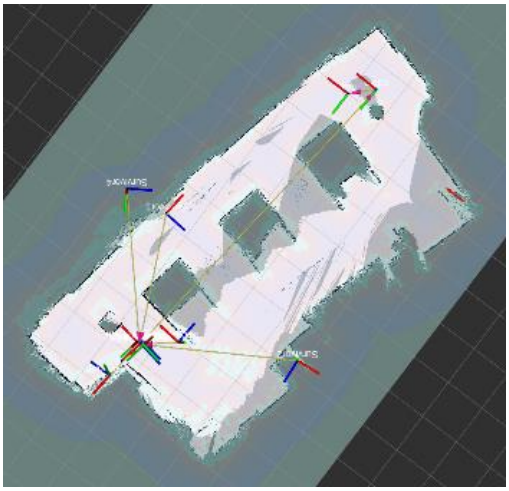

Fig 2. Map generated with off-the-shelves configuration

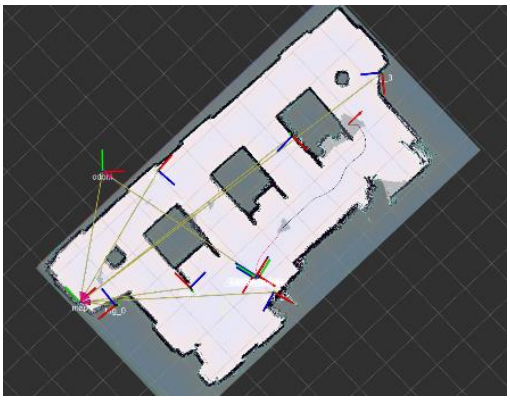

Fig 3. Map with tuned SLAM and frontier exploration

### C. *Detected AprilTags*

The off-the-shelf system uses the base level apriltag_ros package to detect AprilTags, whereas the modified system additionally performs averaging of the detected pose estimates to increase the accuracy of the pose on the occupancy grid map. It can be seen on Fig 2 that some AprilTag poses are marked out of the map, and some poses are marked far off of the real position in the test environment. However, the occupancy grid map generated by the modified system in Fig 3 shows a near-perfect detection of all the AprilTags with their poses on the map.

## IV. Conclusion

Disaster response scenarios present a challenge in which responders may have to endanger themselves to locate victims. This is where an autonomous system becomes very useful, as information can be gathered about an environment without endangering rescue teams. Our proposed system uses a Turtlebot3 Burger equipped with a camera and LiDAR along with software components used for exploration, SLAM, navigation, and detection of victims. The system used SLAM Toolbox for SLAM operations, explore_lite for frontier exploration, move_base for navigation, and apriltag_ros for AprilTag detection. The system also used custom configurations and software to better adapt the system to the task at hand (e.g., the agglomeration of AprilTag poses or the generation of a masked occupancy grid map). In the end, the proposed system does very well against the benchmark created using only off-the-shelf components (i.e., no custom configurations or software). The proposed system creates a more accurate map of the environment and estimates the poses of "victims" more accurately as well. In all, the proposed system would be a valuable asset in a search and rescue scenario.

## V. Future Work

The modified design worked very well in exploring a simulated lab space and detecting AprilTags. However, the system design could still be further refined to more accurately locate and pinpoint victims (AprilTags). The main idea that was proposed to accomplish this was to design a way for the robot to orient itself in a more optimal position, or a series of optimal positions, that would allow for it to better record the pose of the AprilTag.

As the system stands, the robot will search the environment and attempt to scan the whole map with its camera. This ensures that the robot sees everything in the environment. However, this does not ensure that the robot will have been in an optimal position to record the pose of an AprilTag; it just ensures that it has seen every AprilTag at least once.

The proposed solution to this problem is to create an intermediary node that would listen to explore_lite and move_base and forward the proper messages to either node (e.g, goals, results, etc). If no AprilTag has been seen, goals from explore_lite will continue to be sent to move_base and the system will persist as if nothing had changed. Once an AprilTag is detected from the camera, however, the node would determine optimal positions for viewing and recording the location of the AprilTag. It would then stop sending goals from explore_lite to move_base and instead send move_base the optimal positions as goals. It would continue to do this until there are no more entries in its queue of optimal positions. Lastly, it would mark the AprilTag as fully explored so that it will know it does not need to explore it again the next time it sees the tag. After all of this, the node would resume communication between explore_lite and move_base until it sees another tag or finishes exploring the environment.

This solution would allow for a more accurate recording of AprilTag locations. However, as the system stands, the robot will likely find itself in many valid locations to record the AprilTag, and it will agglomerate all of the recordings, so it will already get a decently accurate recording of the location of the AprilTag. This solution would become useful if the robot happens to see an AprilTag from far away and never needs to move closer to it in order to complete its exploration. With the proposed solution, it will actively try to get a better reading regardless of if it has explored that portion of the environment or not.

## VI. References


[1] "move_base," [Online]. Available: http://wiki.ros.org/move_base. [Accessed 2022].

[2] S. Macenski, "SLAM Toolbox: SLAM for the dynamic world," *Journal of Open Source Software,* 2021.

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

[4] "AprilTag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 4193-4198.

[5] S. M. a. A. S. a. B. K. a. M. A. Abbas, "Analysis and Improvements in AprilTag Based State Estimation," *Sensors,* vol. 19, 2019.