Autonomous Ground Robot For Reconnaissance using RRT Exploration

Akhil Sasi Kumar, Gaurav Mishra, Laleth Indirani Nehrukumar,
Shirish Kishore Kumar, Venkatesh Desai, Yash Mewada
Team Splinter
Northeastern University

***Abstract*—Miniature robots for use in reconnaissance missions are beneficial, considering their uninvolved hardware design and their task-specific software customizations. Rescue missions can be aided by first obtaining a preliminary analysis of the environment from the bot and then executing human deployment. In this paper, we deployed an autonomous reconnaissance-aimed system that consists of 2-wheeled *differential drive robot*. In it, we have considered our key design focus to be the optimization of the environment mapping algorithm. As the algorithm manages the localization and mapping procedure, our primary task entails the detection of April tags(substitutes for victims) scattered in the environment. The *TurtleBot* is stocked with a 360° LiDAR for localization and mapping and a Raspberry Pi Camera to detect the tags. The *gmapping* based *SLAM* and the Rapidly exploring Random Tree (RRT) based environment exploration are the respective ROS packages tasked with mapping and exploring.**

***Index Terms*—Autonomous Navigation, Robotics, Robotic Reconnaissance, RRT Exploration (Rapidly exploring Random Tree), PRM Exploration (Probablistic Roadmaps), Robot Operating System (ROS).**

I. Introduction and Motivation

A. Background

Being able to autonomously navigate through the whole environment is necessary for a reconnaissance bot to generate a map and locate victims. There are several reasons why autonomous navigation is useful in a reconnaissance robot in a disaster environment. Firstly, it allows the robot to explore and gather information about the environment without the need for a human operator. This can be useful in situations where it is not safe for humans to be present, such as in a disaster zone. Secondly, autonomous navigation enables the robot to quickly and efficiently explore a large area, which can be useful for search and rescue operations.

B. Problem Formulation

There are many pivotal aspects that are to be optimized during the bot's development. In this project, we recommend the optimization of the environment exploration algorithm with the aim of maximally detecting victims randomly scattered in a test environment. The Turtlebot, a 2-wheeled robot is equipped with a 360° Laser Distance Sensor LDS-01 and a camera with a 62° field of view deployed in an unknown terrain. The objectives are to create a complete map of the unknown environment and identify victim's pose in the map with respect to the origin.

C. Approach

We have sectioned our proposal into 3 divisions:
1. SLAM
2. Exploration
3. AprilTag detection

II. Environment Setup

We have created two environment setups to test RRT-based frontier exploration with 8 AprilTags in each environment.

Environment 1 - 5 obstacles
Environment 2 - 6 obstacles

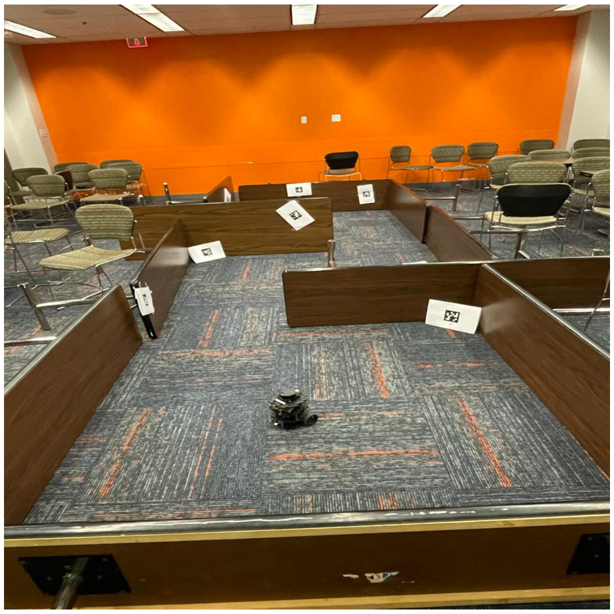


Fig. 1. Environment 1

III. Design Process

A. Simultaneous Localization and Mapping (SLAM)

For the given problem we have utilized the default package "*gmapping*" for simultaneous localization and mapping (SLAM). It is capable of real-time localization and mapping. *gmapping* uses RBPF (Rao-Blackwellized Particle Filter) for localization. We are using 200 particles in our *gmapping* for a better likelihood of the position of the robot with respect

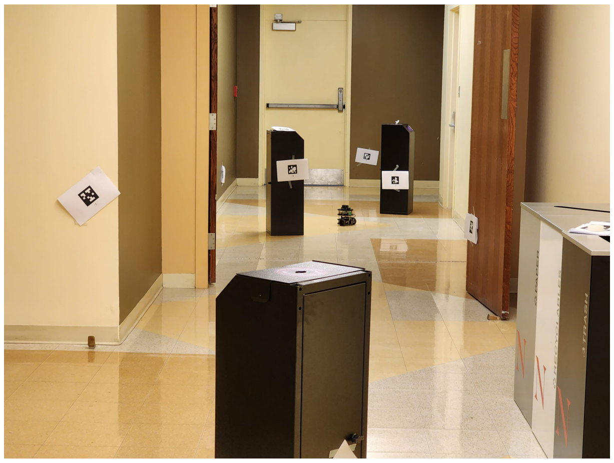
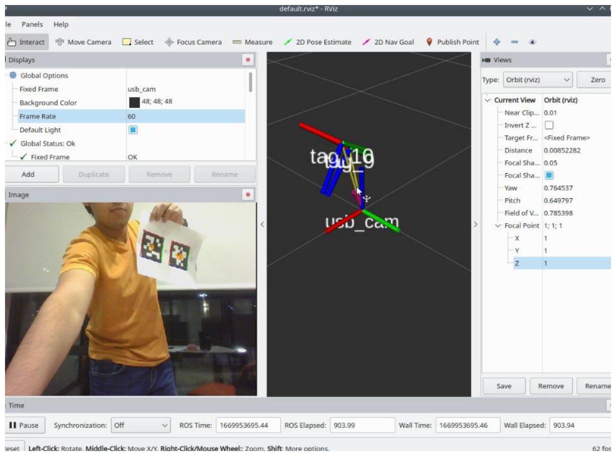


Fig. 2. Environment 2



Global Options
Fixed Frame: usb_cam
Background Color: 48; 48; 48
Frame Rate: 60
Default Light: ☑
Global Status: Ok
Fixed Frame: OK

Type: Orbit (rviz)
Current View: Orbit (rviz)
Near Clip...: 0.01
Invert Z ...: ☐
Target Fr...: <Fixed Frame>
Distance: 0.00852282
Focal Sha...: 0.05
Focal Sha...: ☑
Yaw: 0.764537
Pitch: 0.649797
Field of V...: 0.785398
Focal Point: 1; 1; 1
X: 1
Y: 1
Z: 1

Synchronization: Off ROS Time: 1669953695.44 ROS Elapsed: 903.99 Wall Time: 1669953695.46 Wall Elapsed: 903.94

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options. 62 fps

Fig. 3. Visualizing in RVIZ

to the map. It is a lightweight algorithm that requires less computational power and works well in less complex and not-so-big areas. This *gmapping* package is used to create an occupancy grid map of the environment.

B. Pose Estimation

One of the tasks of the reconnaissance bot is to estimate the poses of victims or distressed entities. We use AprilTags to represent the victims. Several AprilTags are placed in various places, orientations, and heights in the simulated environment to accurately represent victims of a disaster.

AprilTags: We are detecting AprilTags using camera frames captured by Raspberry Pi Camera in combination with a ROS package "apriltag_ros"(A ROS wrapper of the April-Tag 3 visual fiducial detection algorithm). The apriltag_ros is defined by the two configuration files config/tags.yaml and config/settings.yaml. These files specify the family of tags, the IDs of all tags to expect, as well as the physical sizes of the tags in our environment. We are using "36h11" family tags and our id's range from 0 to 10. We have used two different set of AprilTags whose physical size are 0.08m and 0.172m.

The apriltag_ros subscribes to two topics one for camera frame (*/camera_rect/image_rect*) and other for camera's intrinsic calibration (*/camera_rect/camera_info*). We calibrated our Raspberry pi camera using opencv camera calibration package. We used a 9x7 checkerboard to calibrate and got a yaml file containing Camera matrix, Distortion, Rectification, and Projection data.

The apriltag_ros package publishes the topics */tf, /tag_detections and /tag_detections_image*. The pose is returned in the quaternion form and we use a *scipy* function to convert it into Euclidean. The transformation from the Apriltags frame to the camera frame is represented by T_{CA} and the transformation from the origin of the map to the LIDAR frame is T_{OC}. The following formula has been used to compute the pose of AprilTags in the world frame, represented by T_{OA}.

$$T_{OA} = T_{OC}.T_{CA} \tag{1}$$

Algorithm – 1 : Pseudocode for storing Apriltag pose w.r.t origin

```
rospy.Subscriber("/tag_detections", AprilTagDetectionArray, get_tag_detection)

$T_{OC}$ ← tf_listner('base_scan','map')                    #Pose of lidar with respect to origin
tags ← {}
detected ← []
if length(tag_msg.detections) ≠ 0 then
    $T_{CA}$ ← tag_pose ←tag_msg.detections.pose.pose.pose
    tag_id = tag_msg.detections.id
    if tag_id not in detected:
        detected.append(tag_id)
        $T_{OA}$ ← np.matmul($T_{OC}$, $T_{CA}$)
        tags[tag_id] ← $T_{OA}$
    end if
end if
```

Fig. 4. Pseudocode for Storing AprilTag pose

For this project, we have assumed that the distance between LiDAR and the optical center of the camera is not significant since we expect the results of the Turtlebot exploration are transmitted to a rescue mission team. The team would then use the pose information received from the Turtlebot to actually bail out the victims.

IV. Proposed Solution

A. Exploration

We are using the RRT-based exploration [4] [5] algorithm for exploration. The algorithm generates a tree-like data structure in the environment, with branches representing possible paths that the robot can traverse to the frontiers it detected. It then randomly samples points from the array of centroids and tries to connect them to the tree, using local and global planning algorithms to determine the feasibility of each path (a Depth First Search approach). This process continues until the robot reaches its destination or the tree covers the entire space. *In RRT exploration the sequence of points defined before the exploration begins plays a vital role. RRT is heavily biased towards unexplored regions and is "complete",

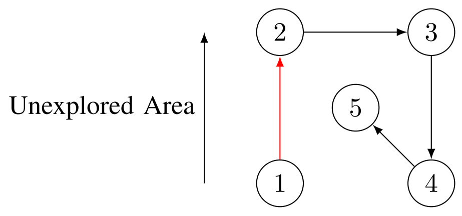


Fig. 5. Correct Sequence of Exploration points where 1 is the start point/robots origin

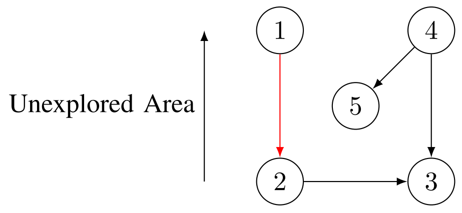


Fig. 6. Wrong Sequence of Exploration points where 2 is the start point robots origin

which means that it will always find a feasible path from the start to the goal if one exists. Therefore, RRTs are useful for autonomous navigation in complex environments because they can quickly find a feasible path even in cluttered or unpredictable situations. However, the completeness of RRTs comes at the cost of potentially generating a large number of paths and taking a long time to run.

We are using the package "rrt_exploration" to implement RRT-based exploration in our Turtle bot [6]. This package was developed for ROS Kinetic, but we are porting to use it in ROS Noetic and this has caused us a few issues such as in one case even though the frontiers were found and the command was sent accordingly to /move_base, the dwa_planner was unable to execute it.

Why sequence of points matter? In using the RRT-based exploration method, sequence of points play an important role because they define the progression in which the robot needs to explore the unknown environment. RRT needs 5 points to define this sequence. It always considers to explore in the direction from the point 1 to 2. The first 4 points are the vertices of the bounding box in which the turtlebot is expected to explore the unknown environment (as shown in Fig 5). Therefore, the user defining these points must have the birdeye view of the environment he wants the turtlebot to explore. The fifth point is defined near the starting point of the turtlebot to close the loop.

The slam_gmapping node takes in sensor *_msgs/LaserScan* messages and builds a map *nav_msgs/OccupancyGrid* using the 360° LiDAR sensor values.

In order to use rrt_exploration the move_base_node node, which brings up the navigation stack on the robot, must be running. This package (rrt_exploration) generates target

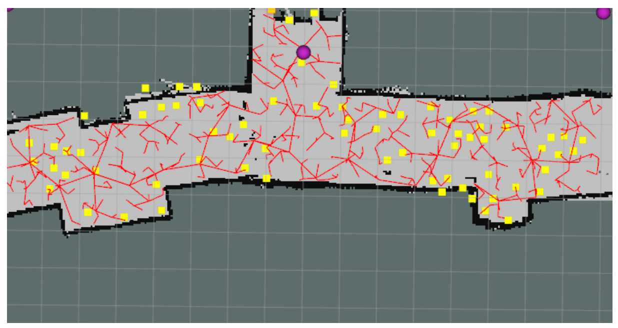


Fig. 7. Map created by using rrt_exploration and gmapping

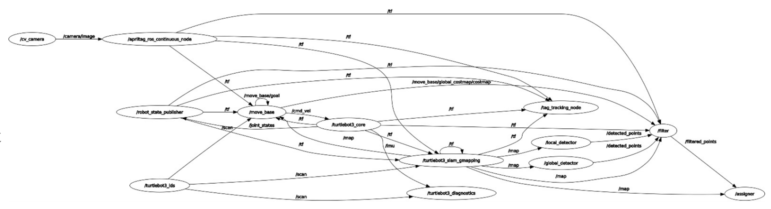


Fig. 8. Node graph

exploration goals LiDAR values, the robot must be able to receive these points and move towards them. Additionally, robots must have a global and local cost map. All of these are provided from the move_base_node. There are 3 types of nodes, nodes for detecting frontier points, a node for filtering the detected points, and a node for assigning the points to the robots [6]. By doing this the robot creates goals inside explored regions. This helps us because the more time turtlebot spends in the unknown environment, the more AprilTags it detects that may have been missed before, and the accuracy of the map also increases.

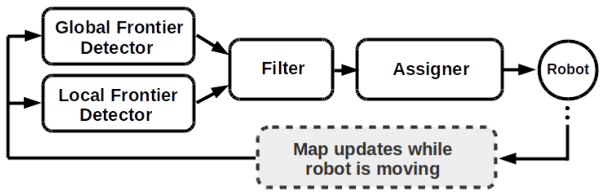


Fig. 9. Global and Local frontier detection in RRT-exploration

This method ensures that an accurate map is produced and simultaneously increases the chance of detecting an AprilTag. Fig. 9 shows the flow graph of nodes, topics, subscribers, and publishers used in order to achieve this functionality.

Github Code:https://github.com/venkydesai/Team-Splinter-Autonomous-Ground-Robot-For-Reconnaissance.git

V. Results

The deliverables of this project was to generate an occupancy grid map of an unknown area with obstacles in it and the pose and tagID of the detected apriltags in the environment. To test the robustness and repeatability of our approach we set up two environments, one with enclosed area and another one with open environment and in both cases, the quality of the map was exactly the same. Also in both environments, two kinds of Apriltags were deployed, one with 0.08m (environment 1) and another one with 0.172m (environment 2) and in both the scenarios the robot was able to detect 7 out of 8 deployed tags. It was not able to detect the tags which were at a significant height compared to the robot.

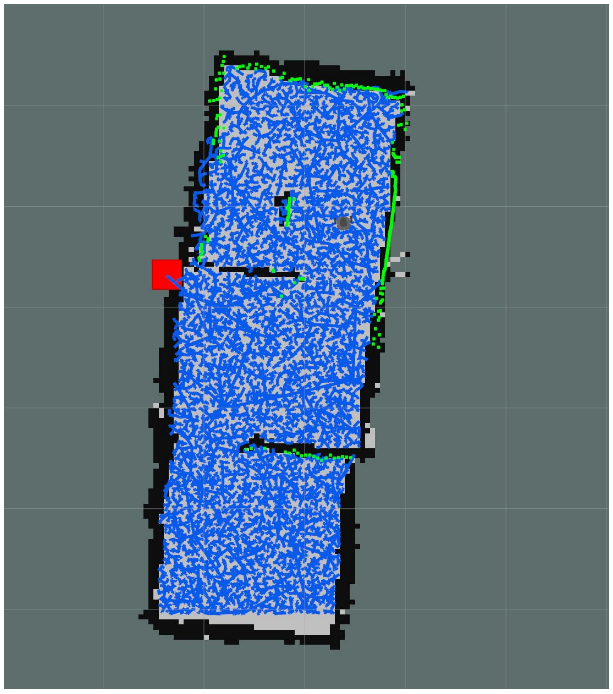


Fig. 10. Complete Map of environment 1

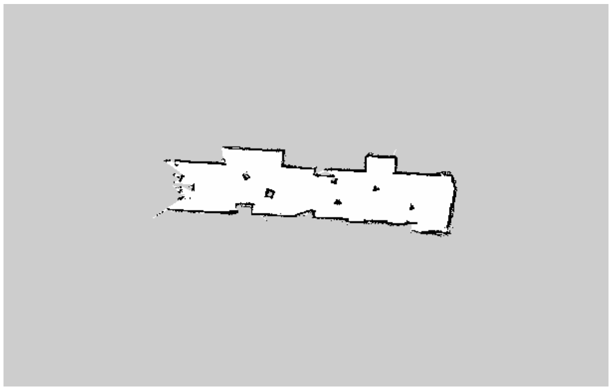


Fig. 11. Complete Map of environment 2

Below is the output we got for one of the pose of apriltag.

$$T_{OA_6} = \begin{bmatrix} -0.26 & -0.96 & -0.07 & -25.58 \\ -0.89 & 0.27 & -0.34 & 19.42 \\ 0.34 & -0.02 & -0.93 & 5.92 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

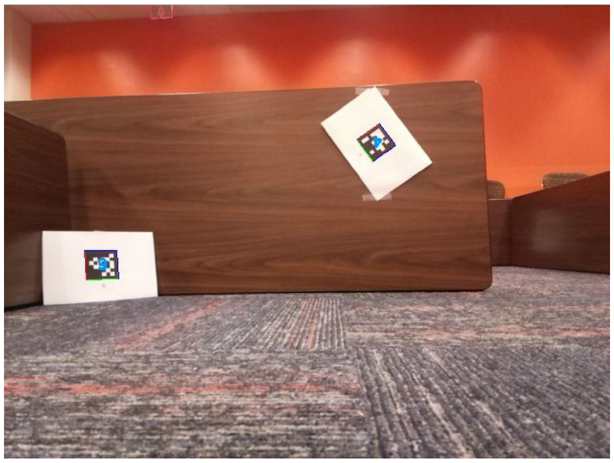


Fig. 12. Detected AprilTags with respect to robot frame

A. Benchmark results with off the shelf exploration algorithms

Our main objective was to create an occupancy grid map of the environment and detect apriltags in the environment. We used a simple ROS package fro apriltag detection hence no bench marking was needed for that. Whereas a different approach was taken to autonomously explore the map. We tried the explore lite package in gazebo which did not guaranteed a complete map generation.

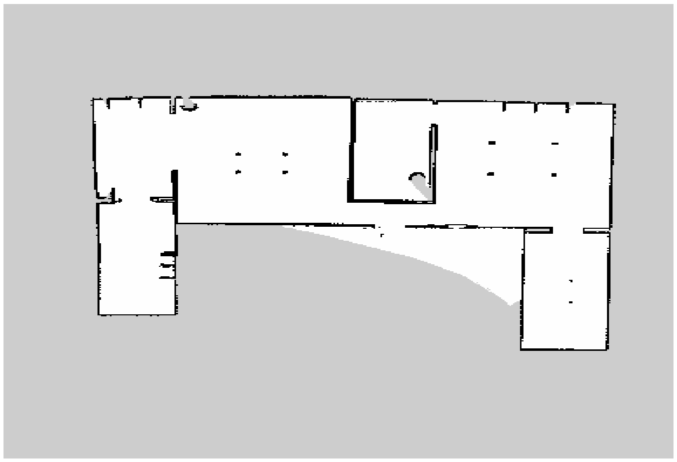


Fig. 13 Explore lite package Gazebo simulation

VI. Conclusion and Future Work

Our proposal maintained a primary objective of optimizing the maximal detection of the Apriltags scattered in the environment. This was enabled in 2 parts. Firstly, our RRT-based environment exploration algorithm simultaneously localized the robot and explored the areas around it and produced an entire occupancy grid of the environment. Secondly, the Apriltag detection node was able to accurately capture the pose information about the detected tags in the environment. As we mentioned above about the possible issues occurring while incorporating RRT-based exploration, we can see in the results above that there were instances where the robot was not able to accurately localize itself due to the less optimized algorithm for both RRT-based frontier exploration and publishing the pose of the tags with respect to the world frame.

The optimization of robot localization and mapping problem along with the pose of apriltags detected is left for future work. A wider approach that can be taken into consideration for such scenarios would be to explore the possibilities of PRM (Probabilistic Road-maps) based frontier exploration due to fact that such sampling based planners assures maximum travel-ability which helps in a more precise occupancy map generation and better object of interest detection/recognition.

References

[1] https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[2] https://github.com/AprilRobotics/apriltag_ros

[3] http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

[4] H. Umari and S. Mukhopadhyay, "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 1396-1402, doi: 10.1109/IROS.2017.8202319.

[5] Gurel, Canberk Suat, Rajendran Sathyam, Rajendra Mayavan Guha, Akash. (2018). ROS-based Path Planning for Turtlebot Robot using Rapidly Exploring Random Trees (RRT*).

[6] https://github.com/prabinrath/rrt-exploration

[7] https://docs.opencv.org/4.x/dc/dbb/tutorial-py-calibration.html