Northeastern University

360 Huntington Ave, Boston, MA 02115

Master of Science in Robotics
Team: Donatello

Report on

Frontier Exploration based Autonomous System for a Simulated Disaster Environment

Submitted by:

Vignesh Ravikumar
Skanda Akkihebbal Prasanna
Pavan Rathnakar Shetty
Santhosh Sankar
Aadhar Bansal

Submitted To:

David Rosen
Assistant Professor
College of Engineering

Contents

Github Repository:

https://github.com/Skanda-sap/Project_donatello.git

Video Link of robot moving:
https://youtu.be/w4cfwrwOQMQ

Rviz recording:
https://youtu.be/8T0anl1iNOs

Summary

In this project, a complete autonomous system to perform reconnaissance in a simulated disaster environment has been implemented using a Turtlebot3 mobile robot. Onboard the robot, we use a Raspberry Pi 3 with the Robot Operating System (ROS) framework for Simultaneous Localization and Mapping. **move_base** package from the ROS navigation stack has been used to command the robot to move around the environment. Lidar scanners present on the robot were used to generate occupancy grid maps using the **GMapping** package. **explore-lite** frontier exploration package was used to explore the environment. Raspberry-Pi camera was used to detect the Apriltags which are assumed to be the hostages in the disaster environment using the **apriltag_ros** package. Lookup transforms were used to estimate the position of the Apriltag with respect to map frame.

System Construction and Rationale

TurtleBot3: TurtleBot is a ROS standard platform robot. It contains a raspberry pi for computation and a OpenCR microcontroller for directional drive. The reason we chose to use a TurtleBot3 is that it has necessary ROS packages and support. We can install ubuntu on the raspberrypi and SSH into the system directly from our remote PC for control. LIDAR sensors, raspberrypi camera are connected to the Turtlebot3 for input.

move_base: The move_base package provides an implementation of an action that, given a goal in the world, will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task. The move_base node also maintains two costmaps, one for the global planner, and one for a local planner that are used to accomplish navigation tasks.
The reason we chose to use move_base is that this component is the mainstay of the navigation algorithm. move_base is a package which integrates various aspects of navigation, each of which adheres to a standard API defined in the ROS standard. This standard interface allows the user to easily implement their own algorithms for each aspect of robotic navigation (Local and Global planning, Recovery, Costmaps).

Mapping:

We need to map the disaster environment to get an idea of how the environment is, and for moving the robot around.

The **gmapping** package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called slam_gmapping. Using slam_gmapping, you can create a 2-D occupancy grid map from laser and pose data collected by a mobile

robot.

The reason we chose to use Gmapping is that it is one of the most common systems for robot application. The system uses Particle filter and creates the grid-based map. The system subscribes to the sensor msgs/LaserScan message and publishes nav msgs/OccupancyGrid,tf transformation.

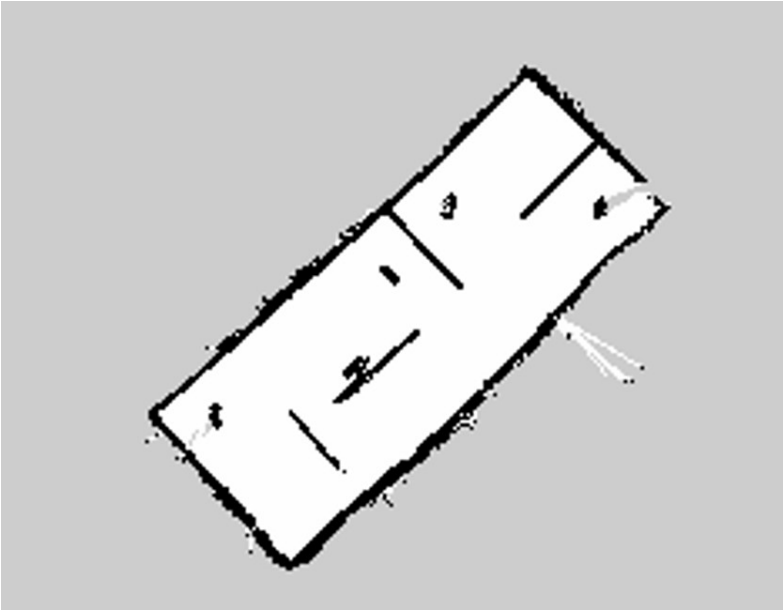

Fig. (1) Map of the environment

Exploration:

We need to explore the disaster environment with the robot to map the environment to understand the scene and find the places where the victims (here, apriltags) are present.

There are several exploration algorithms that are present for mobile robots, starting from Bug0, Bug1 algorithms, wall following algorithms. But these algorithms face a bottleneck when it comes to certain tricky environments and loop into a deadlock.

One interesting exploration algorithm which we came across was Rapidly exploring

Random Tree. Even though the RRTs are constructed incrementally in a way that quickly reduces the expected distance of a randomly chosen point to the tree, we need to manually add the rectangular region around the robot to be defined in the RVIZ window using four points and a starting point for exploration within the known region of the robot. We felt like this is an extra step that is manually done and defeats the purpose of a fully autonomous system.

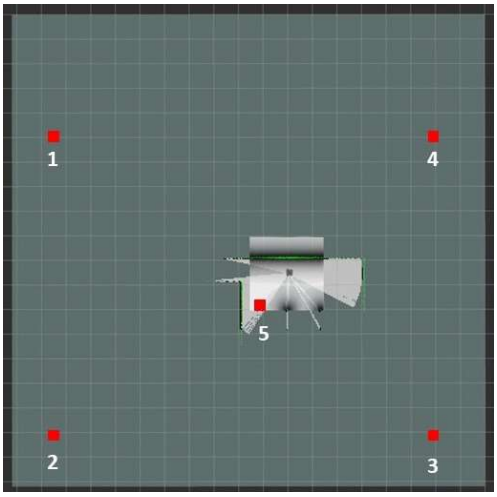

Fig. (2) RRT algorithm rectangular region setup in Rviz

Hence, we looked at frontier-based exploration algorithms like **explore_lite**. Frontier exploration is the most common approach to exploration, wherein frontiers are regions on the boundary between open space and unexplored space. By moving to a new frontier, we can keep building the map of the environment, until there are no new frontiers left to detect. The advantage of this algorithm is that the robot can explore large open spaces as well as small, cluttered spaces.
In ROS it is possible to explore environment with use of occupancy grid frontiers. One of the nodes, that perform this task is explore_server node from frontier_exploration package. This node uses occupancy grid e.g., created

by slam_gmapping and publishes goal position to /move_base/goal topic subscribed by path planner e.g., move_base node.

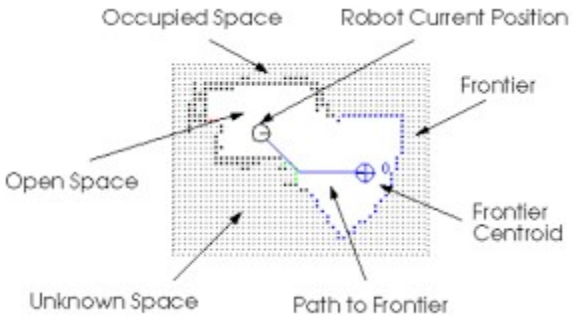

Fig. (3) Frontier Exploration illustration

Apriltags:

Apriltags are an excellent visual fiducial system to mimic an actual victim in a disaster environment. The apriltags can be detected using any camera, and in our project raspberrypi camera that is connected to the TurtleBot3 is used to detect it. We made use of the apriltag_ros package that contains the continuous detection algorithm, and raspicam_node to access the raspberrypi camera with ROS.

Fig. (4) Apriltag 36h11

Pose Estimation:

Pose estimation of the apriltags is important, since that is the entire objective of the project is estimating the location of the victim in the environment. When the robot detects an apriltag with the apriltag_ros package, it only generates the coordinates of apriltag with respect to the camera frame. This is not the correct coordinates, as the map frame's origin are different from the camera frame.

Therefore, we have written a python script called detection.py, that uses lookuptransforms to transform the apriltag pose with respect to the map frame. While running the robot, this script will store all the values and save it in a text file.

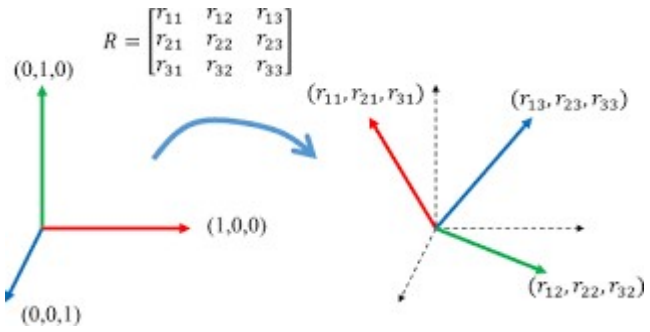

Fig. (5) Illustration of coordinate transformations

Algorithm for coordinate transformation from tag frame to map frame:

1. Start
2. Initialize a node called ‘Node’ and a listener object for TransformerListener()
3. Run a while loop to subscribe to /tag_detections topic
   3.1 The subscribed data is passed as argument to a callback function
   3.2 Run a for loop to do the following:
      3.2.1 Extract the tag_name from the subscribed data
      3.2.2 Using listener.lookuptransform, transform the frames from the tag frame to map frame
      3.2.3 Store the extracted values in a dictionary called dict_apriltag
   3.3 Save the dictionary to a text file for reference
4. Stop

Changing system default parameters for improved efficiency:

- While traversing the environment, there was a bottleneck in the path where the robot could not further explore the environment at one point. We overcame this situation by decreasing the value of *minimum_frontier_size* in the explore_lite function.

```
<launch>
<node pkg="explore_lite" type="explore" respawn="false" name="explore" output="screen">
  <param name="robot_base_frame" value="base_link"/>
  <param name="costmap_topic" value="map"/>
  <param name="costmap_updates_topic" value="map_updates"/>
  <param name="visualize" value="true"/>
  <param name="planner_frequency" value="0.33"/>
  <param name="progress_timeout" value="20.0"/>
  <param name="potential_scale" value="0.001"/>
  <param name="orientation_scale" value="0.0"/>
  <param name="gain_scale" value="1.0"/>
  <param name="transform_tolerance" value="0.3"/>
  <param name="min_frontier_size" value="0.25"/>
</node>
</launch>
```

Fig. (6) Snippet of explore_lite launch file containing different parameters

- *minimum_frontier_size* - Minimum size of the frontier to consider the frontier as the exploration goal. In meters.

  Reducing this value made the algorithm to search for more frontiers, thereby exploring the whole environment. This is important for reconnaissance as there can be victims present in interior environments.
- *track_unknown_space*: true – This is done to use costmap provided by move_base by the explore_lite.

```
shutdown_costmaps: false
controller_frequency: 30.0
planner_patience: 5.0
controller_patience: 5.0
conservative_reset_dist: 3.0
planner_frequency: 10.0
oscillation_timeout: 10.0
oscillation_distance: 0.2
track_unknown_space: true
transform_tolerance: 2
```

Fig. (7) Snippet of move_base parameter values

Experimental Results

A model arena that mimics a disaster environment was created for demonstration of our algorithm as to how well it performs. The arena consisted of 15 Apriltags, each of size 16cm was present, and a series of walls in a maze-like fashion for the robot to explore. The total size of the arena was approximately 30 ft. x 10 ft.

Fig. (8) Simulated Disaster Environment

Occupancy grid mapping:

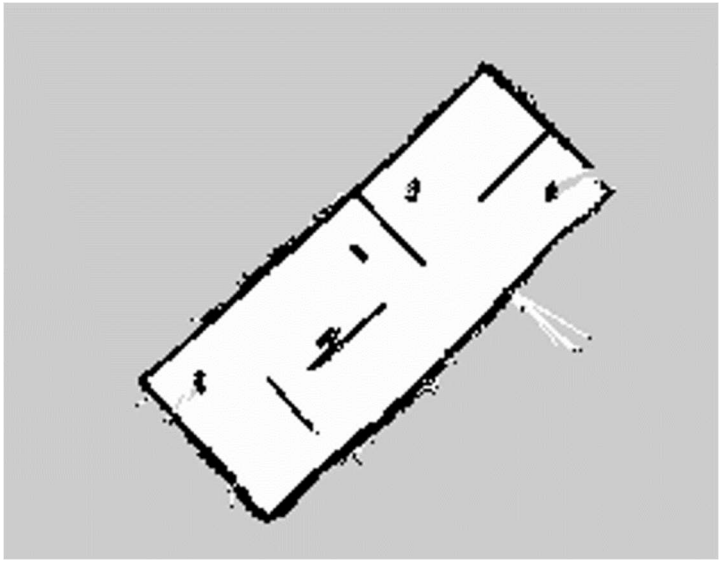

Fig. (9) Map of the environment

The figure above shows the 2D occupancy grid map of the simulated disaster environment that is generated by the Gmapping node.

RQT Graph:

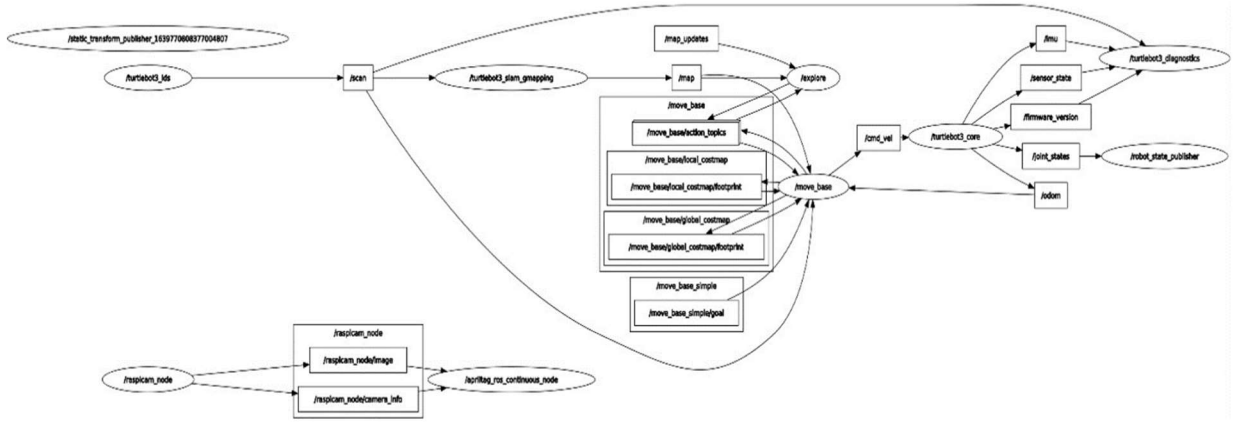

Fig. (10) RQT Graph of the system

The video in the link (https://youtu.be/w4cfwrwOQMQ) above shows the entire trajectory of the robot exploration, mapping, and the number of tags that was detected by the robot.

The robot was able to map the entire arena in under 3 minutes, and was able to detect 13 out of the 15 apriltags.

Conclusion

A robust completely autonomous system that performs reconnaissance in a simulated disaster environment has been implemented. The experiments conducted using the mobile robot such as tuning the exploration frontier values, speed of the robot and their actual effect on a given environment gave lots of insights and visualization of the strength and capabilities of the algorithm working in the backend. There is much scope for development in this project, such as adding a random sampling algorithm that can avoid the bottlenecks and explore the environment thoroughly and find all the victims (apriltags) efficiently.

References:

1. https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/#overview
2. http://wiki.ros.org/move_base
3. http://wiki.ros.org/explore_lite
4. http://wiki.ros.org/apriltag_ros
5. https://wiki.ros.org/noetic/Installation
6. https://github.com/UbiquityRobotics/raspicam_node