EECE 5550 Disaster Response (Rocksteady)

Maulik Patel
Northeastern University
Boston, MA
patel.maul@northeastern.edu

Jarrod Homer
Northeastern University
Boston, MA
homer.j@northeastern.edu

Christopher Swagler
Northeastern University
Boston, MA
swagler.c@northeastern.edu

Connor Nelson
Northeastern University
Boston, MA
nelson.co@northeastern.edu

Aditi Purandare
Northeastern University
Boston, MA
purandare.a@northeastern.edu

***Abstract*—In this project, a TurtleBot robot was used to explore and map an unknown closed environment. Equipped with LIDAR and a camera, the robot collected data as it moved through the environment, using ROS to perform SLAM and generate a real-time map in the form of an occupancy grid. Additionally, a real-time AprilTag detector was developed to detect and track visual markers in the environment. The combination of the AprilTag detector and the SLAM mapping system would enable the robot to navigate the environment accurately and avoid obstacles in real-time. The project demonstrated the effectiveness of the TurtleBot and ROS in exploring and mapping an unknown environment and our implementation of the AprilTag detection shows the potential for computer vision applications such as object recognition and localization in disaster response.**

I. Links

The link to the code used in this project can be found at https://github.com/Mau1ikPatel/EECE5550-TurtleBot. Details regarding the project structure and how to run the code can be found in the README of the repository.

A video demonstration of the final solution can be found at https://youtu.be/6N2YskyDy9I.

II. Introduction and Motivation

Disaster response is extremely important. The ability and speed of a search team to find individuals in an unpredictable environment can be the difference between discovering and rescuing them. Over the last few decades robots have started to take over in this space, particularly in reconnaissance. The ability to deploy and monitor a fleet of mobile robots allows the response teams to explore faster and not expose themselves to dangerous or unpredictable environments. This allows response teams to be overall more efficient and generate action plans with better knowledge of the entire environment.

Our goal with this project was to use the TurtleBot (Figure 1) as an example and develop a system that is able to explore and map an unknown, closed environment and report the locations of AprilTags that were placed in the environment. Specifically, this means the robot will report with a full map of the closed environment using an occupancy grid and then report the location of any AprilTags discovered in the environment as positions on the global map.

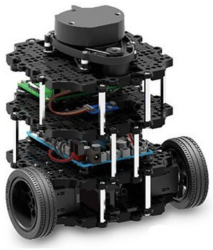

Fig. 1. TurtleBot3 Burger

III. Proposed Solution

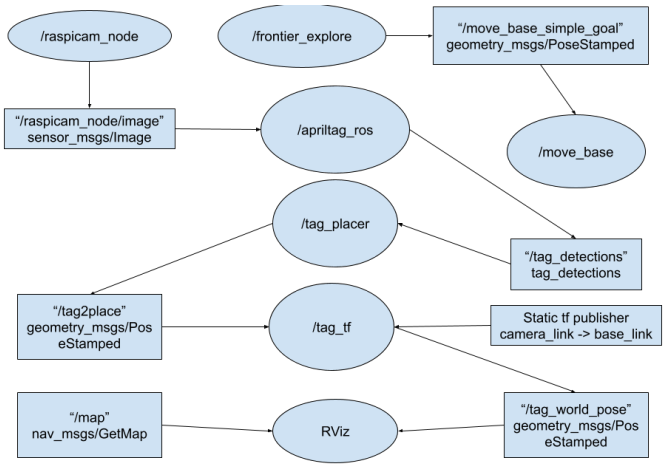

Fig. 2. ROS Node & Topic Diagram

Figure 2 gives an overview of the ROS node and topic architecture. The following sections will explore these in greater detail but from a high level, two ”main” processes run. The first of which is the SLAM and frontier exploration

process. The objective of this process is to generate a map for the robot to understand what its environment looks like and how it is localized within that environment. The second process is the AprilTag detection and localization. The idea is that the robot identifies tags in its environment, and places the location of these tags within the frame of the generated map to understand where these markers exist relative to the robot and within the environment as a whole.

The overall task will be broken down into smaller tasks an explored in further detail:

1) SLAM

SLAM refers to the simultaneous localization and mapping of an environment in robotics used to create a map. The TurtleBot3 was an open-source platform that had built-in SLAM capability with sensors to assess its surroundings such as LiDAR, and RGB-D camera. Using the sensors and the teleoperation node, the TurtleBot navigated around both virtual environments in Gazebo and real-life physical environments, both generating a map of its surroundings and localizing itself within that map. Figures 3, 4, and 5 show the three virtual environments created through Gazebo through which the SLAM node was tested.

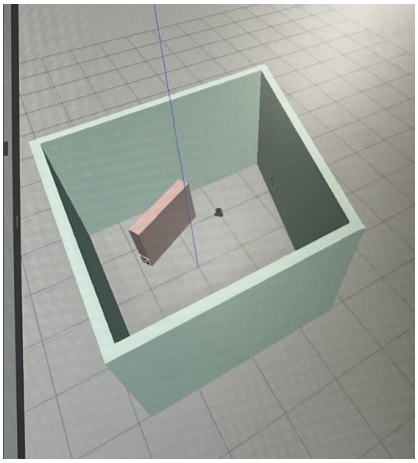

Fig. 3. Basic SLAM Virtual Environment

2) Navigation

The navigation stack utilized by the TurtleBot tutorials is largely reused for navigation. The tutorials were followed in order to run SLAM and also allow for way-points published by RViz to be navigated to.

Autonomous exploration of the environment was based on the wavefront frontier detection. This technique, used in robotics and computer vision, identifies boundaries among varying regions with in a costmap. Here, a wavefront is understood as the boundary between the areas of a map that have been explored versus those that have not. When the slam node, including gmapping, is active the technique identifies the pixels or cells on the costmap that are closest to the robot as a frontier. This frontier is then used by the robot to plan a path of exploration for the unknown environment. In order to decide on which region to explore on the map some parameters are considered such as distance (how close a frontier is), information gain (how much new information about the map can a frontier provide), and accessibility (how accessible are some frontiers for the robot's mobility constraints).

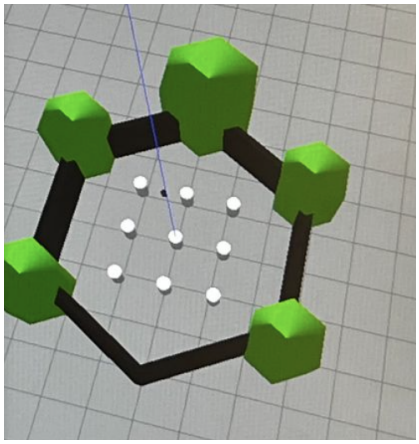

Fig. 4. Default SLAM Virtual Environment

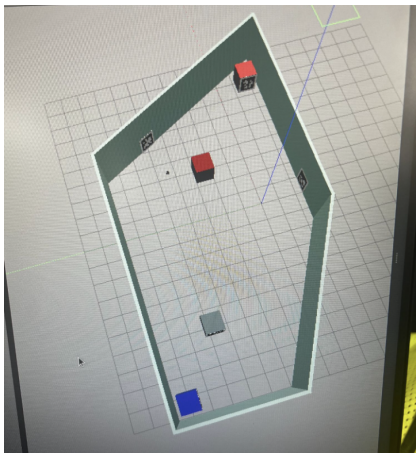

Fig. 5. Large SLAM Virtual Environment

3) Gazebo Virtual Testing

Initially, there were issues on operating the TurtleBot due to running ROS packages on board the TurtleBot with limited RAM, which led to lagging and communication errors. In order to circumvent this issue, virtual testing and simulation were used to tune SLAM and Navigation node parameters before testing on hardware. The first task in this approach was to build 3D environments in Gazebo to allow the TurtleBot to explore. About 6 environments were used in the simulation

testing of the TurtleBot, but only three were worth observing for lessons learned. The Figures 3, 4, and 5 show those environments, ranging as simple, mediocre, and difficult.

To understand whether or not the SLAM, Navigation, and Frontier Exploration parameters were set effectively, there needed to be testing on what a completely explored map would look like. If the map was effectively explored, then the TurtleBot could be sent to search the environment for AprilTags. Hence the SLAM node was ran in each of the virtual environments and a map was created by manually controlling the TurtleBot with the teleoperation node and saved as shown in Figures 6 and 7.

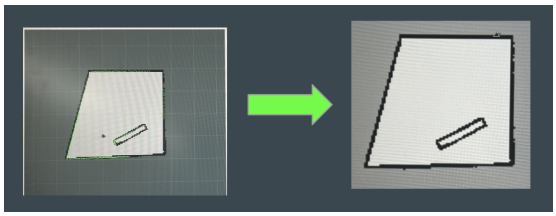

Fig. 6. Map Generated From Basic SLAM Virtual Environment

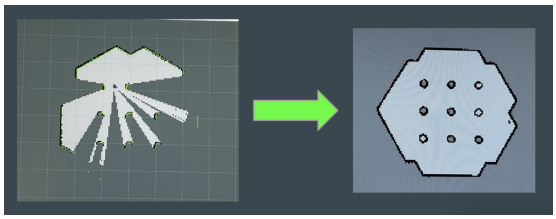

Fig. 7. Map Generated From Default SLAM Virtual Environment

Since processing stress was initially a prominent aspect of the project, SLAM parameters were tuned in order to make sure that the SLAM node was effective in a wide range of environments, such as the virtual ones made in gazebo. To do this, the maxUrange, mapupdateinterval, minimumScore, linearUpdate, and angularUpdate. With maxUrange, the team modified the distance of the LiDAR to allow TurtleBot map a significant portion of the map without a large processing cost. It was also noticed that in particularly large environments without any obstacles in the environment, loop-closure ended up making mistakes in putting the overall map together as shown in Figure 8. It should be also known that each of the virtual environments and physical environments were all mapped using the autonomous frontier exploration method as well. Tele-operation mode was just used for initial testing.

4) AprilTag Detection: Original Approach

Initially, AprilTag detection was carried out using the apriltag library in Python. Specifically, the Detector

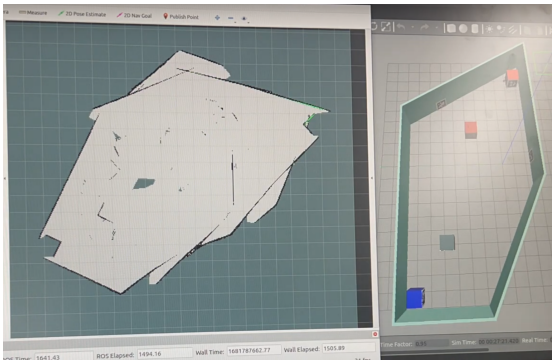

Fig. 8. SLAM Virtual Environment with AprilTags

and DetectorOptions functions were employed. The first implementation was in Google Colab because of the AprilTag/Windows incompatibly. Since external cameras cannot be utilized in Colab, images with AprilTags were uploaded to test. To test detection in real-time, the development environment was switched and a MacOS camera was used. Using the cv2 and apriltag libraries, this implementation successfully identified the AprilTag in frame of the feed by drawing a bounding box around the tag and displaying its ID.

Additionally, to ensure the provided camera was functioning, a Python script outside of the ROS environment was written. Once frames were being captured and written at a pre-determined frequency, the Raspberry Pi camera was verified and functional.

However, the primary limitation of the standalone apriltag library was the lack of pose estimation.

5) Pose Estimation: Original Approach

As pose estimation was not a feature of the apriltag library, a few different pose estimation techniques were explored.

Perspective-n-Point

The Perspective and Point algorithm solves the equation: $[u, v] = KI \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ where u and v represent the image coordinates, K represents the camera calibration, I is an identity matrix, R is the rotation matrix, t is the translation matrix, and [x,y,z] are world coordinates.

This equation is solved for R and t.

The team used cv2.solvePnP [1] to determine the pose. OpenCV solves the aforementioned equation iteratively through Levenberg-Marquarat optimization [2]. In this type of algorithm, least squares is employed to determine R and t that enable a minimum error between the coordinates.

6) AprilTag Detection & Pose Estimation: Final Approach

After dedicating much of the work into manually estimating AprilTag poses, the team discovered apriltag_ros and decided to pivot towards using this singular module instead of splitting between apriltag and a cv2 pose estimation implementation. The beauty of this module is that it not only identifies AprilTags, but also includes pose estimation as part of the published message.
The only necessary configuration for this node was to update the config/tags.yaml file with the tag ID and size. The team printed out AprilTags 0 through 19 with a size length of 0.17 meters. The config/settings.yaml defaulted to using the 36h11 family and publishing the tf, which were both necessary.
Aside from the actual AprilTag configuration, this module also required the correct camera info topic and camera feed topic. In order to get the raspicam node to publish the camera info, it was first calibrated using the camera_calibration's cameracalibrator.py script. After specifying the checkerboard dimensions (8x6) and size (80 mm), this script simply required repositioning the checkerboard at several positions and angles within the camera feed. Once calibrated, the apriltag_ros node ran smoothly, using the camera's calibrated values to correct for any distortions.

7) AprilTag Frame Conversion
With the apriltag_ros node, the messages generated from the tag detection provide information about the ID of the tag and the pose of the tag relative to the camera frame. The pose estimation is only useful for this project when converted into the frame of the map. In order to do this frame conversion, first the tf's static_transform_publisher node was used. This makes the camera frame usable by tf since it provides a reference point to the base link, which is already part of the map's tf tree.
The team developed two scripts in a custom package to do this transformation. The first of which subscribes to the tag_detections topic and converts the given AprilTag message into a PoseStamped message. The second script subscribes to the previous script's topic and converts the PoseStamped message (in the camera link frame) into the map frame which can is used to visualize the pose in the map frame.

IV. Results

In order to evaluate the fully integrated system, an environment was created with four unique AprilTags (IDs 0-3). Outside of the EECE capstone lab, a simple environment was created with irregular walls. Each of the AprilTags were placed on a wall within the environment. The TurtleBot was placed arbitrarily within the environment to begin exploration.

The main assumptions made when creating this environment were that it should be fully enclosed so that the TurtleBot can recognize when the map is complete; that each of the AprilTags in the environment were unique and no duplicates were used; and that the AprilTags were placed only on walls around the same height as the camera and not on the floor.

A complete map of the environment was made using frontier_explore in conjunction with gmapping. After a complete map was made, the robot was moved via keyboard as a proof of concept and to demonstrate the AprilTag detection working. It was observed that AprilTag pose estimates drastically improved after allowing a full map to be generated. The autonomous exploration algorithm was able to build a mostly complete map of the environment, although it missed one small edge. AprilTag detection in the environment was then tested and worked with accuracy of less that 10cm from our estimates. A autonomous navigation algorithm should be implemented in the future to expand on the work completed. Finally, additional visualization of the AprilTags and a well packaged final output would be useful to users.

A link to a video demonstration is found in the Links section at the start of this report.

V. Conclusion

Overall, the work achieved at the end of the project is in line with the group's goals. Utilizing a complete map and placing tags only after already having finishing mapping the environment led to better results. If this were to be extended, a simple state machine could be used. In the first state, frontier exploration would be used to create a complete map of the room. One the map was sufficiently finished, the state machine would transition to the searching phase and explore the map. Instead of using remote control on the robot, an algorithm could be used to generate a path for the robot to take. There are several possible approaches. One approach would be to generate random points across the map and transverse to them. This would not guarantee camera coverage of all of the walls though. A better method might be to use wall following to travel the entire perimeter at a set offset. Tag positions were more accurate when the robot was stationary. A method to stop the robot if a tag is detected to get a stationary estimate could provide better tag pose estimations.

VI. Contributions

A. Maulik Patel

Created and ran virtual environments with built in April tags in Gazebo to test and tune SLAM, Navigation Node, and Frontier Exploration parameters. For true autonomous navigation in unexplored environment, implemented Wavefront Frontier exploration on gazebo and carried over to TurtleBot in a physical environment. Worked on april tag detection and transforms to place april tags on RVIZ map.

B. Jarrod Homer

Conducted initial research on ROS architecture, libraries and packages that could be used for the project. Worked on TurtleBot bring-up, tutorials, SLAM and Navigation. Attempted to utilize explore_lite and apriltag_ros packages. Later helped get apriltag_ros working and used the tf2 library.

C. Christopher Swagler

Configured the TurtleBot, installing libraries, dependencies, and configuring the network. Worked on TurtleBot bring-up, AprilTag-ROS integration, and final environment testing.

D. Connor Nelson

Worked on initial TurtleBot setup, ROS configuration, custom ROS package creation, camera calibration, and real-time AprilTag detection.

E. Aditi Purandare

Worked on static AprilTag detection, pose estimation approaches, early ROS setup research, initial script to test camera, early SLAM research: RRT+gmapping, and early SLAM setup on TurtleBot. Helped with calculations/debugging apriltag_ros pose detection.

REFERENCES

[1] https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html
[2] https://learnopencv.com/head-pose-estimation-using-opencv-and-dlib/