Frontier Based Exploration for Map Building

Tianyi Gu, Zhuo Xu

Fig. 1: Turtlebot2

***Abstract*— In this project, we implement the frontier-based exploration algorithm combined with the occupancy grid mapping technique that enables a Turtlebot robot to autonomously build a map for an unknown environment. The theory of Bayesian inference has been applied to update an occupancy map and the frontier based exploration algorithm has been applied to navigate robot to unknown areas in the map. The experiment results show that the robot is able to map the environment with fully autonomous both in simulation and real world environments.**

I. Introduction

In this project, we implement the frontier based exploration algorithm[5] and the occupancy grid mapping[4] technique that allows a robot to efficiently explore an unknown environment and build a map autonomously. We use a Turtlebot2 robot (Figure 1) in this project.

More specifically, our implementation allows the robot to scan the environment, build a map, identify the obstacles and frontiers in the map, and continue explore those unknown areas until all the areas are fully explored and mapped. Several mapping tasks are tested in both simulation and real world environments. The experiment results show that the robot is able to map the environments with fully autonomous. Discoveries, analysis and explanation will be discussed in discussion section.

II. Methodology

In this project, we apply frontier exploration algorithm on a Turtlebot robot to automatically make an occupancy map of the environment. We implement the algorithms in ROS system. There are two sub-modules, *make map* and *frontier exploration*, in our project. Each of them is a ROS node in the system. In this section, we first talk about our ROS architecture then talk about the two modules.

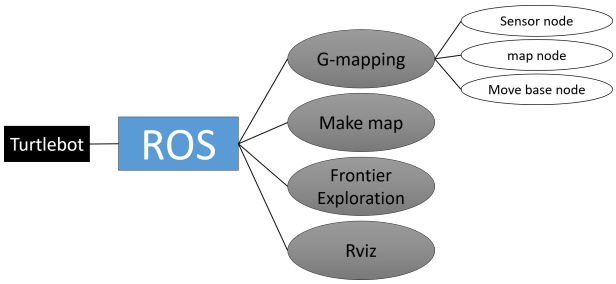


Fig. 2: The ROS system architecture

A. Architecture

In this project, We used TurtleBot2 robot with Kobuki base and Kinect camera. The system is running in the ROS Indigo distribution within Ubuntu 14.04 OS. Figure 2 shows the system architecture. The system would first bring up the robot by the command specified in the turtlebot tutorial document[1]:

```
> roslaunch turtlebot_bringup
minimal.launch
```

We then bring up 6 different ROS nodes: 3D sensor node, gmapping node, move base node, make map node,and frontier exploration node. The first three nodes would be bring up by the gmapping launch file:

```
> roslaunch turtlebot_navigation
gmapping_demo.launch
```

Then we bring up the two algorithm nodes:

```
> rosrun tgu_project makeMap
> rosrun tgu_project frontierExploration
```

We finally run the rviz launch file that brings up the rviz visualizer with our saved configuration.

```
> roslaunch tgu_project rviz.launch
```

B. Make Map

In the make map node, an occupancy grid map is updated and published over time. Figure 3 shows an occupancy map generated by the make map node. Figure 4 shows the subscriber and publisher topics of the make map node. It

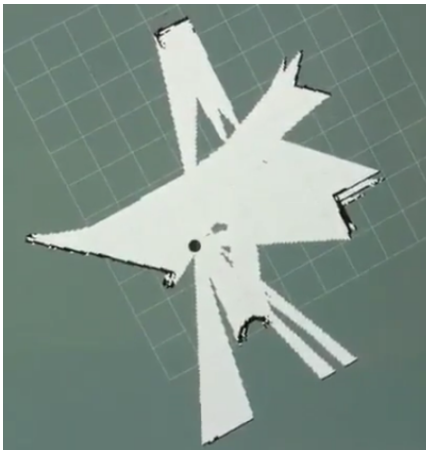


Fig. 3: An occupancy map generated by make map node

Fig. 4: The information flow of make map node

subscribes to the topic *scan* to get the laser scan information. The scan here is translated from the depth camera data. The 3Dsensor node, which bring up by the gmapping launch file, would handles that translation. In our project, we get the robot pose information by subscribing to *odom* topic. The gmapping node also provide a transformation that transform robot pose into *map* frame. Although we can also get the pose by reading the *TF* topic, we found that our result would be a lot messed up because the robot pose is jumping a lot. This is because the estimated robot pose is updated overtime in gmapping. Thus we decided to use the *odom* pose information. Make map node publish the occupancy map to *frontier_map* topic.

Algorithm 1 UpdateMap

```
scanArray ← UpdateScan()
pose ← UpdatePose()
for each scan in scanArray do
    cell ← GetCellOf(scan,pose)
    UpdateObstacleCell(cell)
    UpdateLineOfSight(scan,pose)
```

Algorithm 2 UpdateObstacleCell

```
Cell.Prob ← BayesUpdate()
```

Algorithm 1- 3 are the pseudo code of make map algorithm. In Algorithm 1, given a scan and the robot pose, we can transform the scan into world coordinate system and further get the related occupancy map grid cell. Then we apply Bayesian update in Algorithm 2 to update the occupancy probability of the obstacle cell. We use the 0.9 as the true positive sensor model $P(z|x)$ and 0.2 as the false positive sensor model $P(z|\neg x)$ in the Bayesian update. In Algorithm 3, we apply the Bresenham's line algorithm [2] to determines the cells in line of sight in between the robot and the obstacle cell. Then we update those free cell by a low occupy probability. In our experiment below, we set the number as 0.3.

Algorithm 3 UpdateLineOfSight

```
Cells ← GetLineOfSight(scan,pose)
for each cell in cells do
    Cell.Prob ← FreeCellProb
```

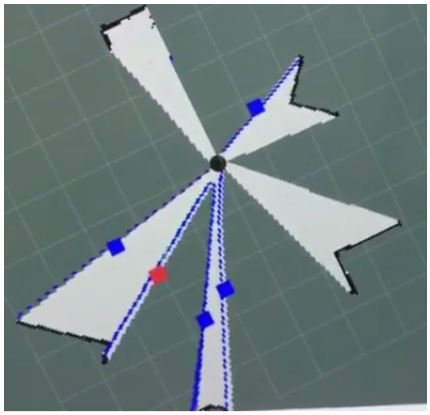


Fig. 5: The information flow of make map node

$$P(x|z) = \frac{P(z|x)P(x)}{P(z|x)P(x) + P(z|\neg x)P(\neg x)}$$

C. Frontier Exploration

Given the current updated map, the frontier exploration node analysis the current frontiers on the map and navigate the robot toward one of them. Figure 5 shows the frontiers and their centroids found by the frontier exploration node. Figure 6 shows the subscriber and publisher topics of the frontier exploration node. It subscribes to the topic *frontier_map* to get the latest occupancy map and to the topic *move_base/status* to check the status of the move base node. This node also subscribes to the *odom* topic to get the robot pose for doing 360 degree rotation at the begin of each iteration.

The frontier exploration node publish to 4 topics. After getting the result of frontier analysis, the node publishes the frontiers and their centroids to *frontier_marker* and *centroid_marker* topic. The node also publishes the best centroid to *move_base_simple/goal* to trigger the move_base node that

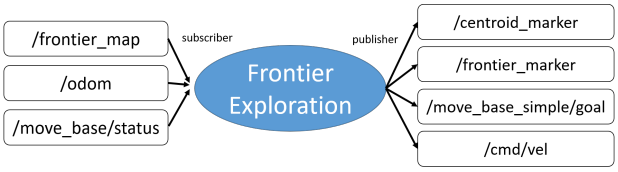


Fig. 6: The information flow of frontier exploration node

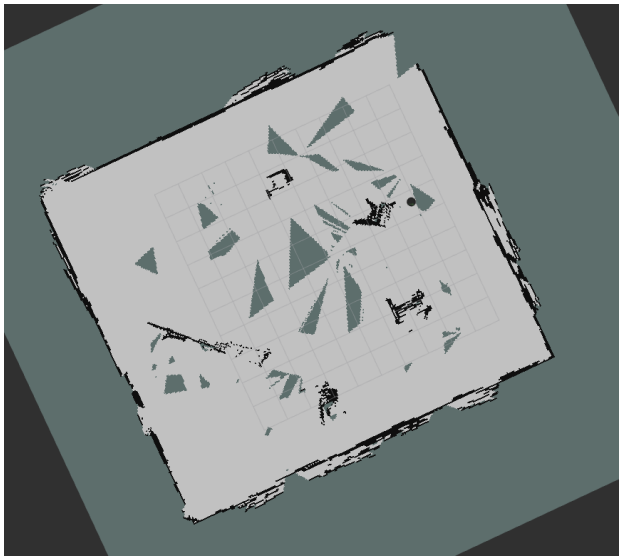


Fig. 7: A map that was generated by make map node in the simulation world

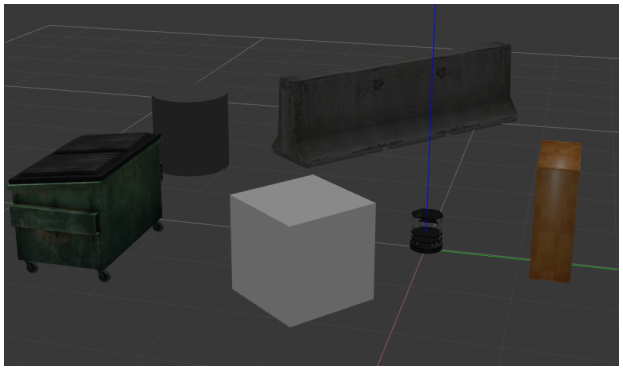


Fig. 8: A simulation world environment in Gazebo.

finds a plan and navigates the robot toward the selected centroid.

Given the occupancy map, we use the connect component labeling algorithm[3] to label the frontiers. We then calculate the centroid of every frontier by averaging over their cell coordinates of frontier cells. To select the best centroid, we apply the following metric to trade off between the frontier length and the distance of centroid from the robot.

$$util = \frac{length}{distance}$$

III. Results

Figure 7 shows a result map generated by the make map node in gazebo simulation world. Figure 8 shows the world. Here we only run the make map node and use key teleoperation to control the robot. As we can see, the built map is a little bit lousy. This is because of the sensor noise of odometry information.

Figure 9 shows the process of building the map in simulation. We use the same simulation world shows in Figure 8. As we can see, the robot is able to build a map from scratch and autonomously navigate it self to the boundaries of built map and unknown areas. The videos of this process can be found here: `https://youtu.be/TExIvkEH9z0` and here: `https://youtu.be/oSZgvkNAuGE`.

Figure 10 shows one cycle of frontier-based exploration algorithm. In panel (a), the Turtlebot completed a 360 degrees rotation and scanned the environment. In panel (b) the frontiers and their centroids are detected and marked as blue lines and cubes; the selected centroid is marked as a red cube. To make the robot behave reliable, we filter out those frontiers that contains less than 20 cells. In panel (c), a trajectory that found by move base node is marked by the green curve. In panel (d), the Turtlebot started moving towards the goal along the trajectory. In panel (e), the Turtlebot keep moving towards the target centroid(red dot) while updating the map. In panel (f), the Turtlebot reached the target centroid(red dot). The robot will repeatedly continue this process until there is no valid frontier.

Figure 11 shows the process of building a map in a real world environment. In this environment, we have an obstacle in the center of the map. The robot is start at the left side of the environment. It first took a 360 degree full scan of the environment and detect two valid frontiers in the map. The move base was able to navigate the robot toward the frontier that is at the left side behind the obstacle. After it arrived the centroid of the selected frontier, it take another full scan of the world that successfully map the whole environment and thus terminate the mapping process. A video of this process can be found here: `https://youtu.be/ItDXtMRaAPw`.

IV. Discussion

Overall, our system woks very well. It runs very stable and can successfully generate maps in simulation. After several fails on real robot testing, we realized that we have to use different parameters in real world. For example, in the simulation, we set the filter parameter as 20 to filer those frontiers that has less then 20 cells. While, in the real world this parameter does not work because no frontier has cells larger than 20 since the world is so small. So we change it to 10. After several tunings, the robot is able to successfully mapping the world automatically.

In real world testing, we found that the task is more likely to fail on day time and always success after sunset. We suspect this is because the depth camera is sensitive to light, and we have a large window in the world environment close to robot. Another reason could be the traffic of network. Because the ssh connections to the turbot heavily rely on the wireless network in Kingsbury Hall, the network could be a lot busy in day time and cause the delay in data transmission thus further cause the robot update the frontiers based on the obsolete data.

V. Conclusion

In this project, we implement an approach to enable a Turtlebot robot to explore an unknown environment autonomously. Our approach applies the Bayesian theory to

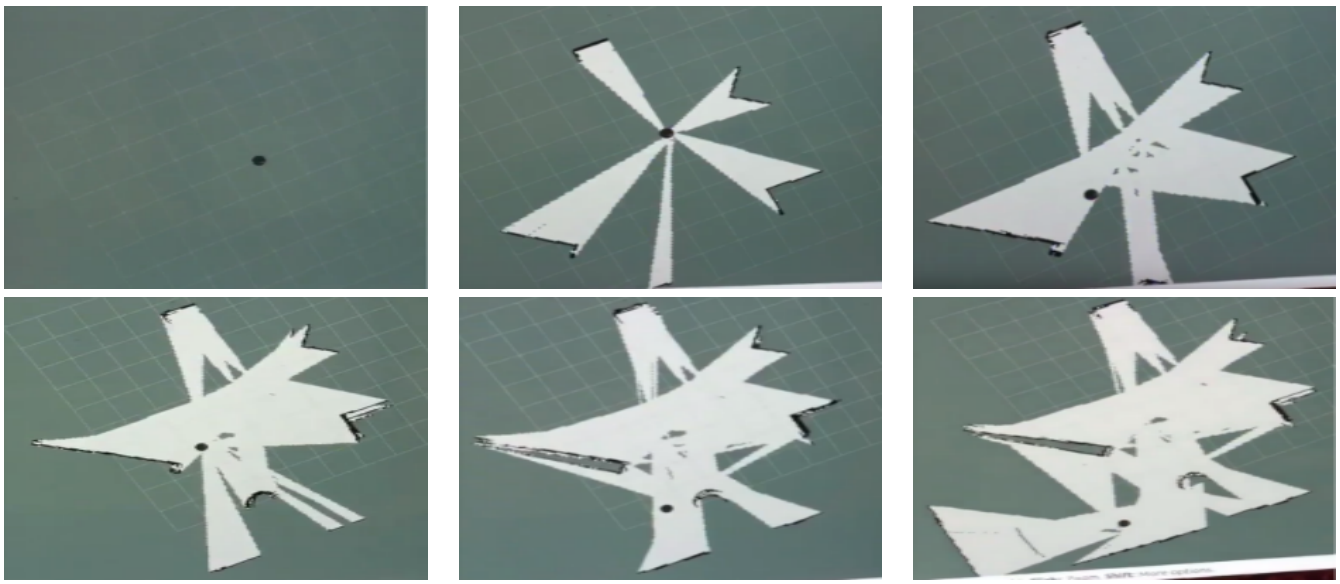
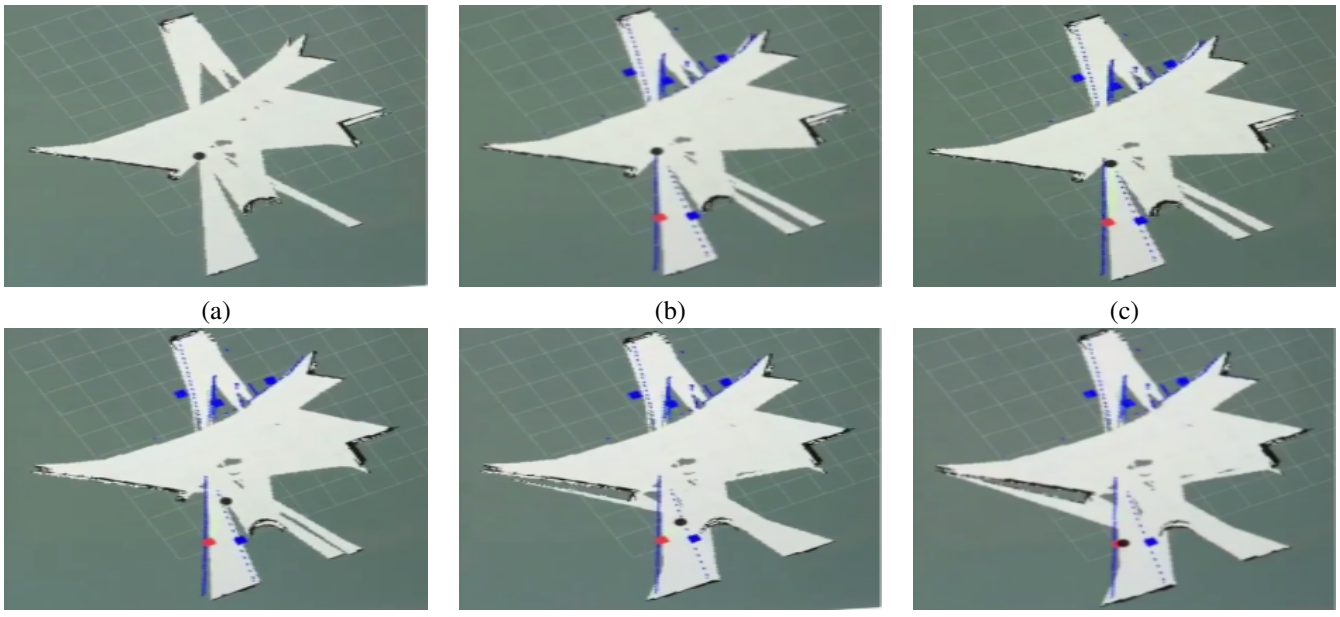


Fig. 9: The process of building a map in simulation world.



(a) (b) (c)

(d) (e) (f)

Fig. 10: One cycle of frontier-based exploration algorithm. (a) The Turtlebot completed a 360 degrees rotation and scanned the environment (b)Frontiers and their centroids are marked as blue lines and cubes; the selected centroid is marked as a red cube (c) The trajectory that found by move base is marked by the green curve (d) The Turtlebot started moving towards the goal along the trajectory (e) The Turtlebot continue moved towards the target centroid(red dot) while updating the map (f) The Turtlebot reached the target centroid(red dot)

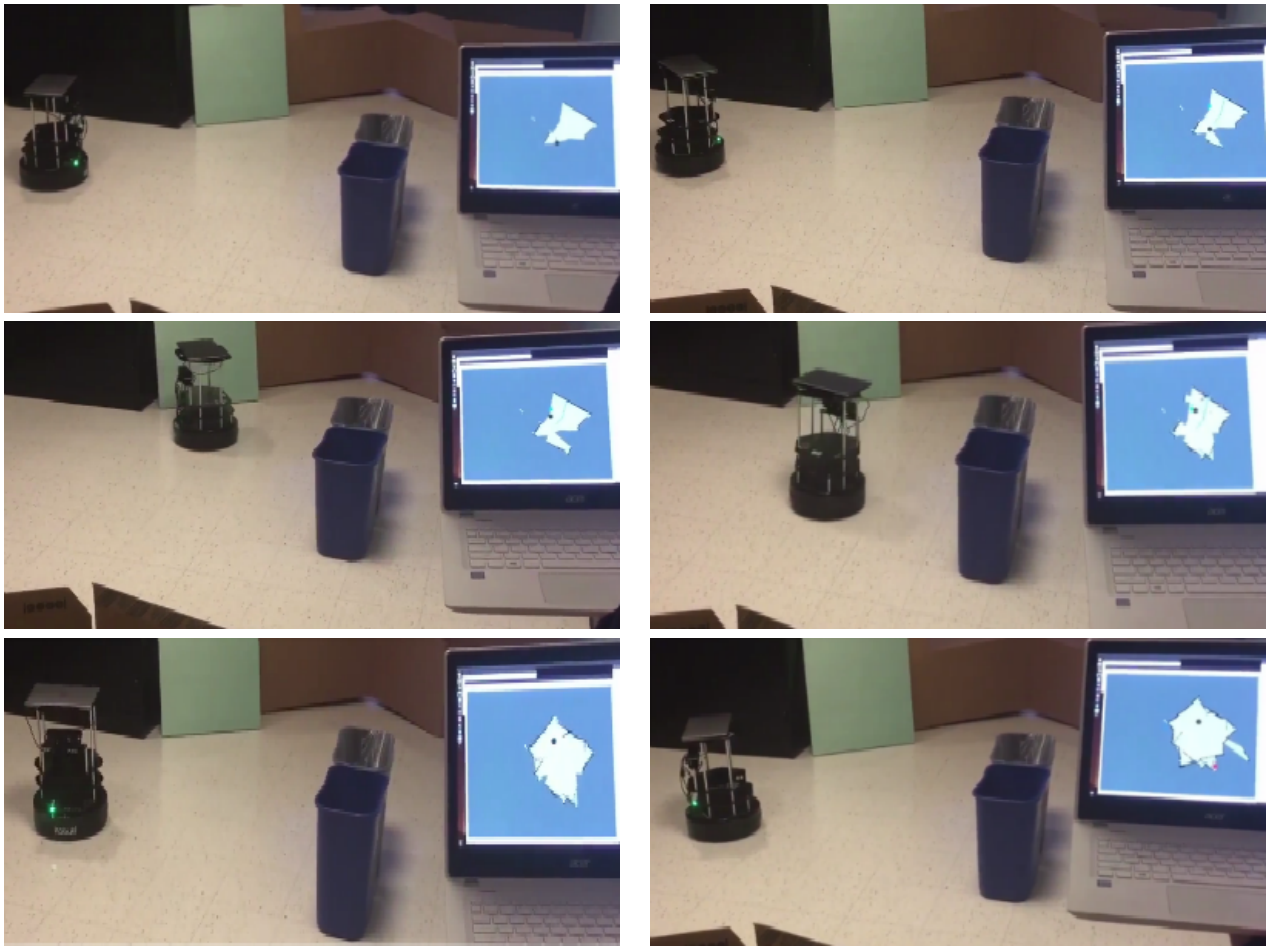


Fig. 11: The process of building a map in a real world environment.

sequentially update an occupancy grid map algorithm and used the frontier based exploration algorithm to navigate the robot toward the centroid of the most promising frontier. The experiment results show that the robot is able to map the environment with fully autonomous both in simulation and real world environments.

REFERENCES

[1] ROS Wiki turtlebot/tutorials/indigo. `http://wiki.ros.org/turtlebot/Tutorials/indigo`. Accessed: 2018-05-10.

[2] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[3] Michael B Dillencourt, Hanan Samet, and Markku Tamminen. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM (JACM)*, 39(2):253–280, 1992.

[4] Hans P Moravec. Sensor fusion in certainty grids for mobile robots. *AI magazine*, 9(2):61, 1988.

[5] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151. IEEE, 1997.