Autonomous Frontier Exploration, Mapping and Path-Planning using Octomap

Anirudh Puligandla, Shubham Wagh, Hassan Saeed and Dousai Nayee Muddin Khan

Abstract— **The project deals with implementing and testing a high-level controller for Turtlebot in a simulated environment wherein the map is unknown. We propose an autonomous exploration strategy for the robot to explore an unknown map as much as possible. This task is accomplished in two stages. Firstly, the robot explores the unknown environment by simultaneously building the octomap of the environment for a certain time. Once the certain time has elapsed, the robot returns to the starting position using this obtained map. These tasks are expected to exhibit more intelligence and robust behaviour as much as possible. The platforms used for the project were ROS for programming and gazebo for simulation. The algorithm was tested on Willow Garage map available on gazebo.**

I. INTRODUCTION

Robots are capable of adapting the work as per the environment sensing by using different algorithms or by using various sensors without any intervention from human being or any objects which can be considered as autonomous moving robots or unmanned robots. Though unmanned robots can be classified into various types like Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs), the way of implementing the algorithm could be much independent. In this paper, we are implementing robot localization, path exploration, path tracking and path planning for turtlebot [1] in gazebo simulating environment based on Robot Operating System (ROS) framework [2], a meta-operating system that supports multiple open-source mobile robot and sensor frameworks.

One of the most important key points of an autonomous robot is its ability to construct spatial models and maps about their environments. Autonomous robots need good exploration strategy to effectively perform the incremental mapping function for partially known areas. There are so many exploration algorithm available under different headings depending on the structure of maps (grid, point, line-based, metric, and topological) and goals of the target selection criteria (costs, earnings).

Anirudh Puligandla, Shubham Wagh, Hassan Saeed and Dousai Nayee Muddin Khan are MSc. VIBOT students in the Escola Politecnica Superior at the Universitat de Girona, Catalunya, Spain.

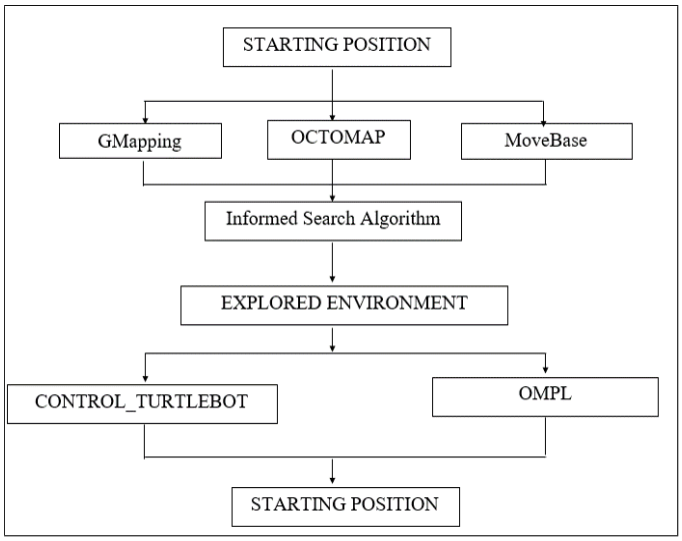


Fig. 1. Implementation Architecture

In this project we propose an autonomous exploration and homing strategy for the robot. This task is subdivided into two stages. Firstly for exploration, we propose an informed search algorithm on a grid, so that the robot could explore the environment. The robot had to be able to locate borders of the unexplored zones and find a path to those borders using an A* search. After that, the robot has to drive to the borders in order to explore those zones by spinning in one place. This task takes place until a certain time of exploration has happened.

For the second task of homing, after a certain time is elapsed we use sampling based algorithm for path planning from the position where the robot has stopped its exploration and with the help of low-level controller, the robot moves towards its starting position.

The paper explains about robot localization and mapping at first for the given unknown environment in gazebo considered as Frontier based exploration

explained in the below section and once we have explored the map in the given time we plan to return the turtlebot to the initial starting position which is briefly explained as return to initial start position in the below section.

In this work, autonomous exploration application has been implemented using frontier based exploration method. In section II, we explain the software overview and different ROS libraries used. In section III, frontier exploration strategy and steps have been described. In section IV, homing steps for the robot have been detailed.. Section V is devoted to limitations in our algorithm and the problems caused due to it. The general conclusions and the future work are given in section VI.

II. Overview

The software component consist of ROS framework and gazebo simulating environment. In gazebo, turtlebot can be actually used just like any other real turtlebot. It consists of the Kobuki base, complete with odometry and contact sensing, and the XBOX 360 Kinect camera for 3D depth mapping. Main benefit of using ROS with a Turtlebot is the availability of simulation and testing packages. For localization, we decided to use open source SLAM (Simultaneous Localizarion and Mapping) libraries. It uses data from the robot's encoders and the camera and probabilisticly estimates the current location with a Kalman Filter.

1) **Laser Scan Matching** - It is a form of visual odometry that is calculated by the help of matching consecutive laser scans of the kinect. The result from this visual odometry is fed into the mapping algorithm for faster convergence.
2) **gMapping** - Simultaneous localization and Mapping (SLAM) task is carried out by gMapping [3] algorithm, utilizing only laser range sensor. With the help of laser scan matching, gMapping can select an optimum initial start point.
   gMapping method uses occupancy grid metric model type maps. Occupancy grid enables fusion of different sensor sources and can run smoothly in high resolution grids. However in large areas as the number of grids increase, calculation cost of gMapping increases dramatically.
3) **Octomap** - It is built using incoming 3D point cloud for scan integration. The *sensor_msgs/PointCloud2* topic is remapped to the sensor data and a tf transform is provided between the sensor data and the static map frame.

III. Exploration Strategy

A. Frontier Detection

Frontier based exploration was first introduced [7]. Frontiers are defined as the border points that happen to lie between explored and unexplored areas which are calculated at the time of mapping and navigation. Navigating to a frontier point enables exploration of unseen areas and adding information to the map. Therefore sequentially advancing to some next frontier defines the frontier based exploration.

After an occupancy grid has been constructed, each cell in the grid is classified by comparing its occupancy probability to the initial (prior) probability assigned to all cells. This algorithm is not particularly sensitive to the specific value of this prior probability [8].

Each cell is placed into one of three classes:

- **empty**: occupancy probability $<$ prior probability.
- **unexplored**: occupancy probability = prior probability.
- **obstacle**: occupancy probability $>$ prior probability.

Let all points that are traversable by a robot in a known map, defined as reachable points. Each reachable point can be defined as neighbor and adjacent, as it is possible to define at least one path connects the robot start point with any reachable point. In partially known maps it is obvious that previously defined paths should pass through the partially known areas. It should also be noted that any path that connects points from partially known area to unexplored areas should pass through a frontier point. Following a path up to several frontiers that connect points from known area to unexplored area each path can be achieved. A robot then can map the entire unexplored area accepting perfect control and sensor information. Frontier based exploration guaranties fully mapping of an unknown environment in finite time but with decreasing amount of marginal information as the explored area expands [9].

```
while(!map_covered&!no_frontier_with_enough_size)
    M ← Current_map
    R ← Robot_position
    F ← Find_frontier_points (M)
    F_c ← Distances_based_connectd_component (F)
    foreach F_c
        F_{c_g} ← Find_cluster_centroid(F_c)
    F_target ← argmin_g{Traversible_distance(F_{c_g},R)
```

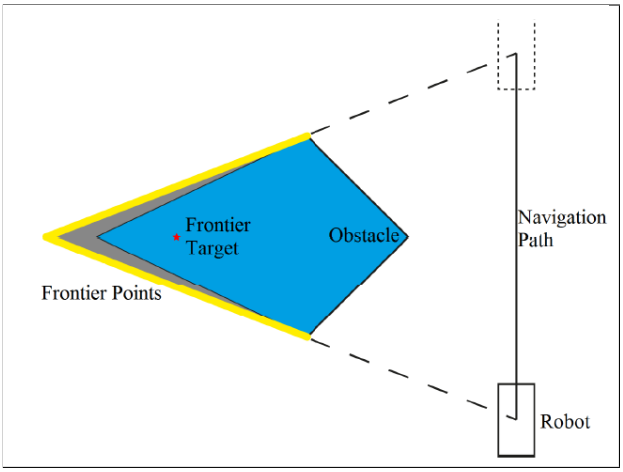


Fig. 2. A state where frontier target is calculated on an obstacle

Candidate frontier points are explored area points with unexplored area point neighbors. In practice it is not feasible to take into account each candidate frontier point. Therefore a point is selected as a candidate frontier point if and only if any 3×3 neighborhood with an explored area point at its center has predefined number of explored and unexplored neighbors but no obstacle points [10].

Frontier points are then clustered with distance based connected component method. Cluster centers are calculated for the clusters that has higher number of points than a predefined cluster size threshold. A cluster center that maximizes some selection criteria is assigned as the current exploration goal.

Exploration goals cannot be always guaranteed to be as reachable before the robot starts its navigation. Such example is given in Fig. 2 where an obstacle shadow intersects to form a triangular frontier cluster (indicated with yellow lines) and the cluster center (indicated with a red dot) is calculated to be on the obstacle (indicated in blue), yet the robot is unaware of the goal being unreachable throughout its given navigation path. Such situations should be taken into account properly at navigation and exploration levels.

B. Navigating to Frontiers

Once frontiers have been detected within a particular evidence grid, the robot attempts to navigate to the nearest accessible, unvisited frontier. The path planner uses a A* search on the grid, starting at the robot's current cell and attempting to take the shortest obstacle-free path to the cell containing the goal location.

1) **A* search :** Cost function is $f(n) = g(n)+h(n)$, $g(n)$ is the cost to get to n from initial node; $h(n)$ is the heuristic estimate of the cost to get to goal from n. In this project, we use Euclidean heuristic, which is a straight line from point n to the goal, thus making it admissible.
2) **Path Generation :** The lowest $g(n)$ values in existing search space from destination to source are backtracked to form a path.

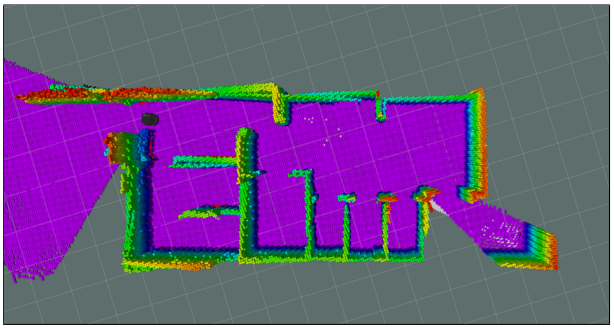


Fig. 3. Octomap generated by robot during frontier exploration

When the robot reaches its destination, that location is added to the list of previously visited frontiers. The robot performs a 360 degree sensor sweep using laser-scan of kinect and adds the new information to the evidence grid. Then the robot detects frontiers present in the updated grid and attempts to navigate to the nearest accessible, unvisited frontier. If the robot is unable to make progress toward its destination, then after a certain amount of time, the robot will determine that the destination in inaccessible, and its location will be added to the list of inaccessible frontiers. The robot will then conduct a sensor sweep, update the evidence grid, and attempt to navigate to the closest remaining accessible, unvisited frontier.

IV. Return to Initial Start Position

After building a map for the given amount of time, the second part the goal was to make the robot go back to the initial position from where the robot has started the mapping process. The start position of the robot can be recorded by obtaining the latest entry from the odometry just before the robot actually starts the exploration and mapping process. The recorded entry can then saved into a text file in any specific format, that can be later read from the same file after the given time expires. For the second part of the goal, once the robot has the start position and the generated octomap, a series of ros services can be called that would activate the turtlebot controller to generate an optimal path from it's current position till the start

position and to provide control commands to the robot to make the robot actually go to the desired position by following the generated path. The path planning for this part of the project was achieved using RRT* algorithm provided in the Open Motion Planning Library (OMPL) library [4]. The code provided used under the last labwork was reused to perform path planning on the generated 2D or 3D octomap. The following paragraphs explain the approach to this problem in detail. The path planning algorithm is also explained in the following paragraphs while the ros services would be explained in later sections.

The OMPL consists of many state-of-the-art sampling-based motion planning algorithm. Any motion planning algorithm provided by the library can be directly integrated with our existing setup as it does not require any particular visualization or collision checking setup. The algorithm generates the path to the provided goal position based on the octomap that has been already generated. The choice of algorithm for path planning for this part of the project is Optimal Rapidly-exploring Random Trees (RRT*). According to [5], RRT* is an asymptotically-optimal incremental sampling-based motion planning algorithm. RRT* algorithm is guaranteed to converge to an optimal solution, while its running time is guaranteed to be a constant factor of the running time of the RRT. RRTs are constructed incrementally in a way that quickly reduces the expected distance of a randomly-chosen point to the tree. RRTs are particularly suited for path planning problems that involve obstacles and differential constraints. Usually, an RRT alone is insufficient to solve a planning problem, but, it does the job for a 2 DOF, differential drive ground vehicle, like turtlebot.

For a general configuration space C, an RRT, that is rooted at configuration q_{init} and has K configurations can be constructed as follows:

BUILD_RRT(q_{init}, K, Δq)
  G.init(q_{init})
  for K = 1 to K
  $q_{rand} \leftarrow$ RAND_CONF()
  $q_{near} \leftarrow NEAREST_VERTEX(q_{rand}, G)$
  $q_{new} \leftarrow NEW_CONF(q_{near}, \Delta q)$
  G.add_vertex(q_{new})
  G.add_edge(q_{near}, q_{new})
RETURN G

Initially, q_{rand} is chosen in C. Next the closest vertex to q_{rand}, x_{near} is selected. Next, a new configuration q_{new} is selected by moving an incremental distance, Δq, from q_{rand}. Finally, a new vertex q_{new} and edge between q_{near} and q_{new} are added and the process is repeated until the goal position is connected to the root.

It can be observed from 4 that the algorithm expands in a few directions to quickly explore the four corners of the square. Although, it can be intuitively expected from the algorithm to randomly explore the space, there is actually a very strong bias towards the places already explored. But RRT* proves to be optimal when using with an existing map with properly defined configuration space.

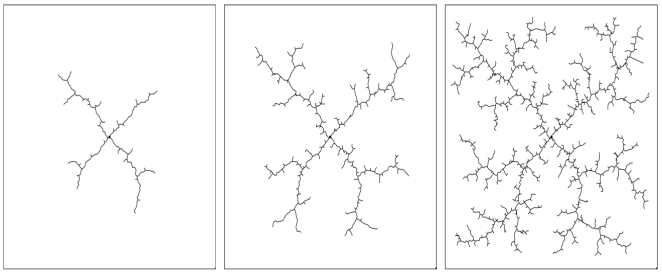


Fig. 4. Construction of RRT for C=100x100, Δq=1 and q_{init}=(50,50)

A. Homing of Turtlebot

As soon as the frontier exploration node starts, in the code we save the initial position of the turtlebot from where it starts its exploration of willow garage environment. A user defined time parameter is defined for the turtlebot to keep on doing map random exploration, until this time limit is reached. After this two ROS services are initiated.

- Start_finding_path
- Ready_to_go

1) Start_finding_path: Once the user defines the Boolean **start** parameter is set **true**, algorithm tries to find the random collision free path to initial position and the query is done via an offline path planner which uses OMPL. In the code the initial pose of the turtlebot is saved in the text file, which is later read to retrieve the starting point location. By calling this service not only we get the path leading to initial location but also the most optimized smooth path leading to the starting position is returned.

2) Ready_to_go: Following user input when *ready_to_go* Boolean is set to **true**, turtlebot starts moving along the path (optimized path) leading towards the initial location of the robot while avoiding collision with the obstacles.

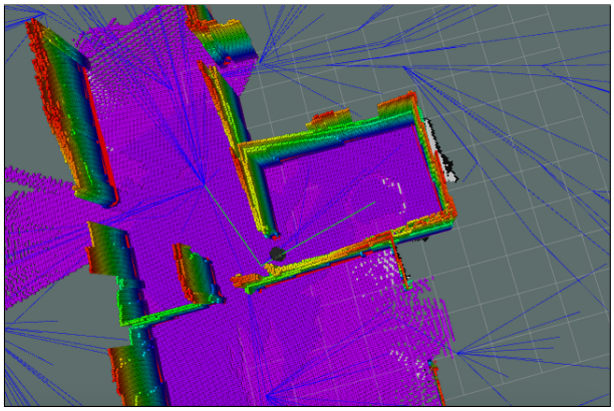


Fig. 5. Blue lines represent different paths to start point; green lines represent optimised smooth path to start point

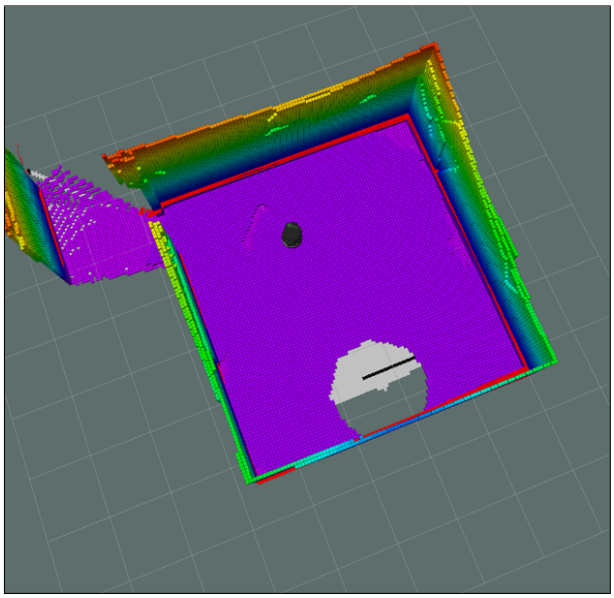


Fig. 6. Limitation of frontier exploration

V. LIMITATIONS

Sometimes when turtlebot starts exploring the environment, it gets stuck inside one room. This is because it doesn't go out of the room until that particular room currently being explored is mapped properly. So a lot of time is invested to properly explore and map a particular room. Following fig. 6 shows an example of such limitation (gap in the Octomap).

VI. CONCLUSIONS

In this paper, we propose an autonomous frontier exploration and homing strategy for any mobile robot. It has been tested using turtlebot on gazebo simulation framework. Even though there are some limitations in the working of the algorithm but the robot was able to successfully explore the environment and return back to its initial start position (homing task).

The project is interesting from the software engineering stand-point because it is very high-level (no low-level robotics involved), allowing to practice search algorithms, such as A*, and performance optimization techniques, such as multi-threading.

In future works, to be able to minimize the revisit to partially explored areas, segmentation based room extraction is considered. As well as, frontier targets can become explored during the navigation. So, moving the robot to the same frontier target can be considered as loss of time. Also, as a future work, it is considered to recalculate new frontier targets when a previous target becomes explored.

ACKNOWLEDGMENT

We would like to thank our Prof. Marc Carreras and Eduard, PhD student for the constant support through out the project and course. We are even thankful to our families and friends.

REFERENCES

[1] Turtlebot Website
[2] ROS Website
[3] G. Grisetti; C. Stachniss.; W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," Robotics, IEEE Transactions on , 23(1):34-46, 2007.
[4] Open Motion Planning Library
[5] S. Karaman and E. Frazzoli, Sampling-based Algorithms for Optimal Motion Planning, International Journal of Robotics Research, Vol 30, No 7, 2011.
[6] More information about RRT
[7] B. Yamauchi, "A Frontier-Based Approach for Autonomous Exploration," in Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on, 1997.
[8] H. H. Gonzales-Banos and J.-C. Latombe, "Navigation strategies for Exploring Indoor Environments," The International Journal of Robotics Research, vol. 21, no. 10-11, pp. 829–848, 2002.
[9] F. Amigoni and V. Caglioti, "An information-based exploration strategy for environment mapping with mobile robots," Robotics and Autonomous Systems, vol. 58, no. 5, pp. 684-699, 2010.
[10] A. Martinez, E. Fernandez. Learning ROS for Robotics Programming, Livery Place 35 Livery Street Birmingham B3 2PB, UK: Packt Publishing Ltd., 2013
[11] Frontier exploration for Turtlebot reference code link