Autonomous exploration report

David Pérez Ruiz
david.perez.ruiz@estudiantat.upc.edu

Fàtima El Baghdadi
fatima.el.baghdadi@estudiantat.upc.edu

Sergio Prieto Molina
sergioprieto.molina@gmail.com

***Abstract*—This paper will explain the different exploration methods to find certain goals in simulation.**

***Index Terms*—Autonomous exploration, cost function, frontier, mobile robot, robot, autonomous robot, goal.**

I. Introduction

In this work two methods for autonomous exploration are presented, both based on the frontier method by maximizing some sort of cost/reward function. One is based on minimizing the cost of the steps the robot has to take to reach the goal, and the other focuses on estimating traffic zones (like doors that open to new spaces) that could yield more valuable information over, say, free areas on the map. The aim of this report is to explain how these methods work exactly, as well as show important improvements on the replanning of the robot and the estimation of the normal to the frontier. Finally, results will be compared to the ones obtained in the last practice for biggest and closest frontiers.

II. The replan criteria

The replan criteria has been modified from the one that was already implemented in the original `exploration_random` to a one that takes into account two replan possibilities: first, if the robot is in the vicinity of the goal but has been wandering around that point for too much time; and second, if the frontier's free center point that is the current robot's goal position changes significantly (above a fine-tuned threshold). If any of the two criteria is met, `replan` is set to true. From the two, the one that has the largest impact to the performance of the robot is by far the second. Being able to update the goal's true position as the frontier changes allows for less loss of time and more accuracy in the movements.
It has to be noted that the replan policy will not be able to overcome path planning issues. If the robot has a goal and the path to that goal can't be defined, the robot will stop moving and replan, but after replanning the same goal as before will be selected and hence the robot will enter a loop.

III. The goal orientation

After a goal at the free center point of a frontier is selected, the orientation that is required for the robot at said point needs to be computed normal to the frontier and aiming at the unknown zone. The way this has been implemented is by doing the following procedure:

1) Compute the vector from the first to the last point of the frontier.
2) Take its normal in both directions. Of course, one will be aiming at the unknown region behind the frontier and the other one at the already-scanned known region.
3) Compute the vector from the frontier center point to the free center point.
4) *If* the dot product of the vector from the frontier center point to the free center point by one of the normal vectors is 0, then they have the same orientation and the orientation of the robot at the goal must be the one of the other normal vector (the one that wasn't in the dot product). *Else*, the orientation of the robot at the goal must be the one of the normal vector that was in the dot product.
5) Set `goal.orientation(yaw)` being the argument `yaw` the arc tangent of the y and x coordinates of the normal vector (whichever direction is chosen).

The steps described above can be easily understood looking at the diagram in figure 1. This algorithm in action can be observed in figure 2.

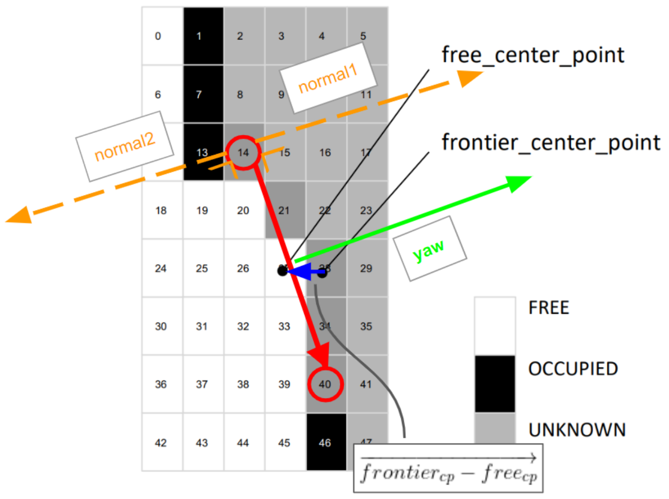

Fig. 1: Diagram that describes the procedure to find the normal of a frontier from its center aiming at the right orientation.

IV. Approach to exploration

We have decided to go for two different frontier-based methods. The first one uses the "Cost-utility function with semantic information" from [1]; and the second one the "Multiobjective

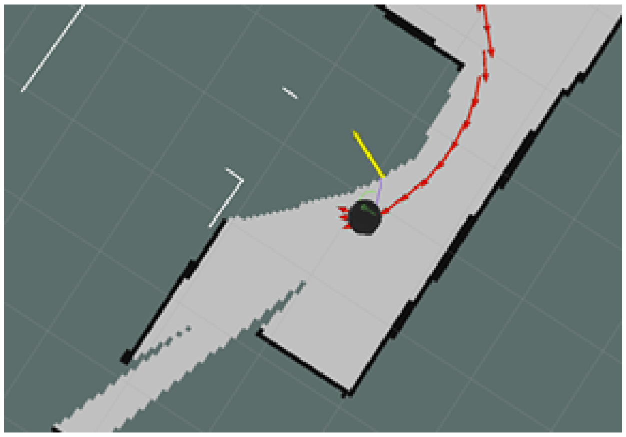

Fig. 2: This snapshot shows how the goal orientation (yellow skinny arrow) is a vector whose direction is normal to this half-moon shaped frontier, aiming at the right orientation (that is, the one looking at the unknown region).

weighted reward function”, which is a combination of biggest-frontier, closest-frontier and the accumulated occupancy cost to reach a goal.

A. *Cost-utility function with semantic information*

The cost-utility method is based on the exploration of the frontier which has the biggest value given by a cost function (equation 1). The cost function takes into account several parameters which allow a faster exploration, one of them (S(F_i)) is related with the concept of traffic area and free area. Traffic areas are the zones in the map that are occluded to the view of the robot and hence, the frontiers that are in contact with the wall, as an example to illustrate this concept, in figure 4 the frontiers with the number 0, 1, 5 are considered as traffic areas. On the other hand, free areas are the areas that are observable by the robot and that were not completely explored by the robot when it was performing a movement. This can be represented in figure 3, frontier number 21, which was not completely scanned by the sensor when the robot was in the starting point (red point near the frontier 21).

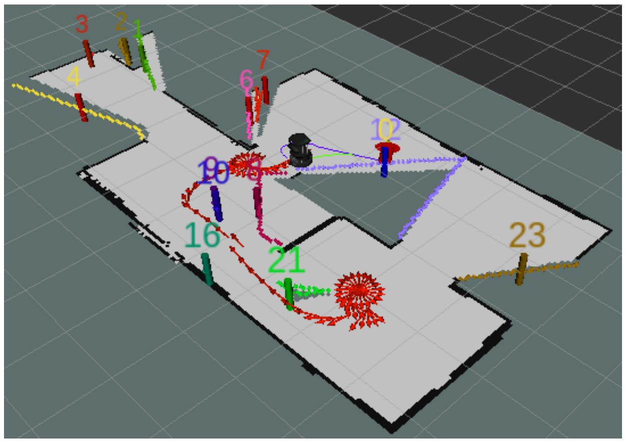

Fig. 3: Free area in simulation

Apart from the consideration of the S(F_i) parameter, the cost function is composed by the following variables.

- F_i
  Concrete frontier where **i** goes from 0 to N.
- **A**(F_i)
  The geometric utility give information about the size of the frontier. Which is used to prioritize bigger frontiers rather than small frontiers.
- **S**(F_i)
  The semantic utility variable corresponds to weather the zone is a traffic area or a free area. In the case that the frontier is a traffic area, the semantic utility is bigger, giving more priority to traffic areas instead of free areas.
- **C**(F_i)
  Topological cost is the distance from the position of the robot to the analyzed frontier. Closer frontiers are more preferable to be explored rather than further frontiers in order to decrease the traveled distance and to speed the exploration.

$$f(F_i) = \underset{F_i}{\operatorname{argmax}}(A(F_i) \cdot S(F_i) \cdot e^{\frac{1}{C(F_i)}}) \tag{1}$$

The implementation of the cost-utility method was formed by several steps. First of all, the frontiers were obtained and for each frontier detected a neighbourhood of 5x5 was analyzed in each extreme of the frontier. Originally, the paper [1] used a camera and a Hough transform to detect vertical lines and therefore traffic areas, however, in our case, the alternative consist in finding in a neighbourhood a value which belongs to a wall. In the case of being connected to an object, the frontier is classified as a traffic area using a value of 80 to the parameter S(p). On the contrary, if any cell in the neighbourhood is not connected to a wall, then the frontier is classified as a free area giving a value of 1 to S(p).
Once the frontier is classified as a free or traffic area, the size of the frontier and the distance from the robot to the frontier are calculated. Hence, parameters A(p), S(p) and C(p) are available to obtain the cost belonging to the frontier.
This cost calculation process is performed in all the frontiers that the robot is detecting. As the aim is to find the maximum cost attributed to a frontier, a search for the maximum is computed. Once found, the robot uses path planning to go to the frontier with maximum cost.

B. *Multiobjective weighted reward function*

The thought process behind this method is: is there a way to combine the biggest and closest frontier methods with the cost of moving the robot through a determined surface? In other words, how can the robot go to the closest and largest frontier but also be penalised for choosing a frontier whose location implies a high displacement cost?[1] The solution is

[1]Let’s take an example: the robot has two possible frontiers in front of it, both the same length, whose center points lie at the same distance. But one requires going next to a wall (which implies big occupancy cost). If a simple biggest-closest frontier method is applied, one frontier will not have prevalence over the other. With this method, the frontier that doesn’t imply going next to a will would be selected as the current goal.

using the `nav_msgs::OccupancyGrid map_` that ROS provides. This vector contains all the cells in which the map is divided in the form of grid, and each cell has a value between 0 and 100 (-1 for unknown cells), being 0 free cell and 100 an obstacle, with all possible values in between (see figure 4 for an example). This "occupancy" information can be leveraged to know the total to reach a goal. The way it is done is by tracing a direct line between the robot and the goal, and then divide it in several steps[2]. For every step the occupancy cost is calculated, so every frontier will have a total occupancy cost. This, and the concept of going to the largest and closest frontier are expressed in formula 2.

$$F_{i\ max} = \underset{F_i}{\operatorname{argmax}}(w_S S(F_i) - w_D D(F_i) - w_O O(F_i)) \quad (2)$$

As it can be observed in 2, the frontier that will be defined as the goal is the one that has the largest cost (or reward, the two words are used interchangeably). The size of the frontier, $S(F_i)$ is the main driving reward, while the distance $D(F_i$ and the total occupancy cost $O(F_i)$ explained earlier subtract from it. The weights w_S, w_D and w_O that multiply them give the user the ability to tune the priority of reducing the occupancy cost and/or distance to frontier, and how important is finding a large frontier. In the simulations performed in this work, values 1, 10 and 1 are used respectively.

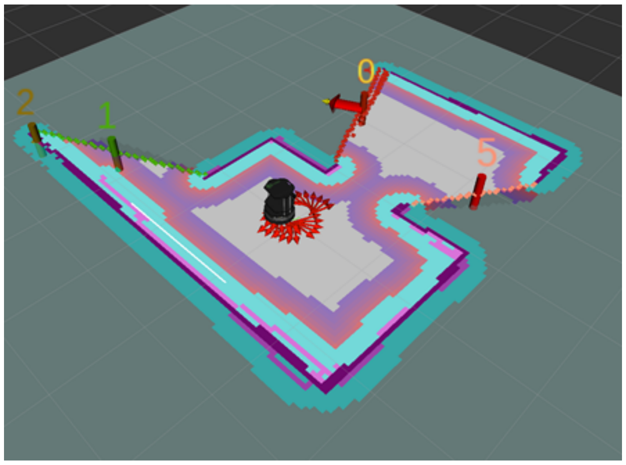

Fig. 4: Example of the cost map superimposed to the grid. The closer the cell is to the wall, the larger the value is.

C. Conclusions

To analyze the performance of the exploration methods developed, traveled distance and area explored were used in the following methods: *Biggest frontier*, *Closest frontier*, *Multiobjective weighted reward method*, and *Cost-function with semantic information method.*

What it can be observed is that the cost-utility method [1] leads to a faster exploration due to the robot explores in less time more area as well as more are with less distance travelled. Hence, the exploration of the map end up faster than the other methods.

Regarding the multi objective weighted method, the velocity to explore the area is between the cost-utility method and the biggest frontier method along with the area explored versus traveled distance. Although using this method makes the robot finish its exploration earlier than the other methods (look at figures 5, 6, where the blue sky line stops sooner than the others), it is not because of the efficiency, but rather because there are errors in the path planing which leads to simulation failures. This has been a recurrent issue all throughout the project. Still, the multi objective weighted method is perhaps too simple of a function: although tunable, it will always look for the path of least effort that is closest and with a largest frontier. The semantic information of the cost-utility method makes it more suitable for efficient exploration.

Nevertheless, both methods presented in this project are able to explore more and in less time in comparison with the biggest frontier method and the closest frontier method. Finally, to conclude, both methods can be tuned in order to achieve a better performance through the modification or addition of several weights in equations 1 and 2, whereas the biggest and closest frontier methods are more rigid in the sense of performance modifications.

[2]The number of steps is arbitrary, but needs not be too small or too large. The important thing is that all distances from robot to goal will be divided by the same number of steps.

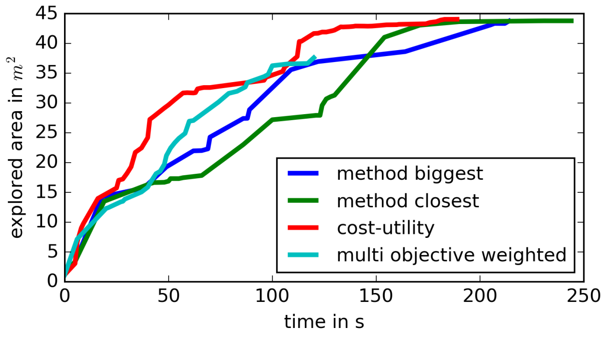

Fig. 5: Comparison of the area traveled versus the time

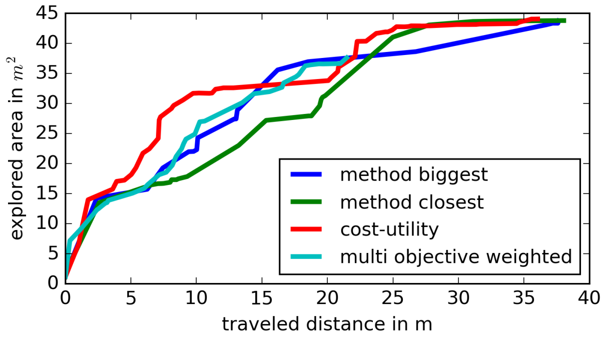

Fig. 6: Comparison of the area traveled versus distance traversed

References

[1] Clara Gomez, Alejandra C. Hernandez, and Ramon Barber. “Topological Frontier-Based Exploration and Map-Building Using Semantic Information”. In: *Sensors* 19.20 (2019), p. 4595. DOI: 10.3390/s19204595.