Autonomous Frontier Based Exploration for Mobile Robots

K. Verbiest(✉), S.A. Berrabah, and E. Colon

Department of Mechanical Engineering, Royal Military Academy,
Av. de la Renaissance 30, 1000 Brussels, Belgium
{kristel.verbiest,sidahmed.berrabah,eric.colon}@rma.ac.be

Abstract. Autonomous exploration of an unknown environment by a mobile robot can be beneficial as robots can navigate in unknown environments without maps being supplied. Also it provides the possibility to produce maps without human interaction. Frontier based exploration is used here as the method for exploration. Some simulation and real world results with a Pioneer 3-AT are presented and discussed. Both simulation and real world experiments use ROS.

1 Introduction

Most mobile robots require some sort of map when navigating and performing tasks in unstructured environments. Usually the territory is mapped in advance by some form of human interaction. As a result, most navigating robots become useless when placed in unknown environments. While many robots can navigate using maps, and some can map what they can see, few can explore autonomously beyond their immediate surroundings. Exploration has the potential to free robots from this limitation. Exploration is defined as the act of moving through an unknown environment and building a map that can be used for subsequent navigation. A good exploration strategy is one that generates a complete or nearly complete map in a reasonable amount of time.

Using ROS and some of its packages, exploration with a Pioneer 3-AT robot is performed in an unknown environment; an overview is given in section 2. The approach used here is based on the detection of frontiers, regions on the border between open space and unexplored space. The methods of the ROS exploration packages employed here are discussed in more depth in section 3. In section 4 some results in simulation and real world experiments are shown.

2 ROS

There are several frameworks available to develop robot software. ROS [1] (Robot Operating System) is run here on the robot and a remote base station overseeing activities. It offers support for various robot hardware platforms and sensors. ROS provides standard operating system services such as hardware abstraction, low-

H. Liu et al. (Eds.): ICIRA 2015, Part III, LNAI 9246, pp. 3–13, 2015.
DOI: 10.1007/978-3-319-22873-0_1

level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It is based on a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages. The library is geared toward a Unix-like system. ros-pkg consist of user contributed packages that implement functionality such as simultaneous localization and mapping, planning, perception, simulation etc.

For mapping the environment the *gmapping* package is used. The *gmapping* package [2] contains a ROS wrapper for GMapping [3]. GMapping is a highly efficient Rao-Blackwellized particle filer to learn grid maps from laser range data. The *gmapping* package provides laser-based SLAM (Simultaneous Localization and Mapping). Using *gmapping*, we can create a 2-D occupancy grid map from laser and pose data collected by a mobile robot. The map is created incrementally and in real-time during the robot motion.

For navigation purposes the *move_base* package [4] provides an implementation to move the robot to a goal point. The *move_base* node links together a global and local planner to accomplish its global navigation task. The global and local planner can be substituted by other planner implementations; they just have to adhere to interfaces laid out in the *move_base* package. The global planner provides a path and the local planner will send appropriate velocity and angular commands to move the robot over the current segment of the global path taking into account incoming sensor data.

For exploration two available ROS packages are used. The *explore* package [5] and the *hector_exploration_planner* [6] package. The *explore* package is a frontier-based exploration library; it generates goals for the *move_base* package. It actively builds a map of its environment by visiting unknown areas. The *hector_exploration_planner* is based on the exploration transform approach presented in [7]. *hector_exploration_planner* provides a planner that generates both goals and associated paths for the exploration of unknown environments.

3 Frontier Exploration

3.1 Method

The goal of exploration is to produce a map of an unknown environment. The method consists out of finding frontiers. Frontiers are the boundary between explored and unexplored space. During exploration, robots navigate to frontiers [8] to extend the known area until the complete (reachable) environment is explored. Which frontier to go to next can be controlled based on the cost of a frontier. The path is planned to the next goal frontier. The robot moves towards this goal autonomously while avoiding obstacles on its way. As the robot continues to move, new information about the environments comes in, leading to the formation of new frontiers. If there are no frontiers left, exploration is complete and the area is mapped.

In both the *explore* package and the *hector_exploration_planner* package occupancy grids are used as an input. In order to get new information, one must go to a fron-

tier that separates known from unknown regions, see figure 1. Such a frontier is a cell in the occupancy grid that is marked as *free* but has a neighboring cell that is marked *unknown*. A segment of adjacent frontier cells is considered as a potential target if it is large enough so that the robot could pass it. If more than one potential target is detected in the occupancy grid, then the frontier which has the lowest cost associated with it is selected.

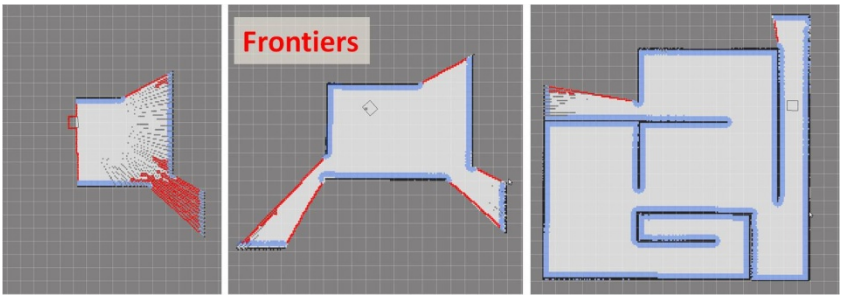

Fig. 1. Screenshots of an exploration missions in the ROS Rviz simulator. The current frontier cells are marked in red.

3.2 Explore package

The explore package returns a list of frontiers, sorted by the planners estimated cost to visit each frontier. The frontiers are weighted by a simple cost function (1), which will prefer frontiers that are large and fast and easy to travel to. The cost of a goal frontier depends on several factors: the distance between the robot and the frontiers, change in orientation to face the frontier and expected information gain of the frontier.

$$cost = weight_1.distance + weight_2.orientationChange - weight_3.frontierSize \quad (1)$$

3.3 Hector_exploration_planner Package

The exploration transform is used in the *hector_exploration_planner* package. Based on the knowledge of the environment that the robot has already acquired, the algorithm calculates a path to the next interesting frontier. This approach takes into account the distance to the next frontier and the difficulty of the path for the robot. Those difficulties can result from narrow passages but also from wide, open spaces where the sensors cannot detect any landmark.

The *Exploration Transform* Ψ of a cell c to reach a target cell c_g is defined as follows:

$$\Psi(\mathrm{c}) = \min_{c_g \in F} \left(\min_{c \in \chi_c^{cg}} \left(l(C) + \alpha \sum_{c_i \in C} c_{danger}(c_i) \right) \right) \tag{2}$$

$$c_{danger}(c_i) = \begin{cases} (d_{min} - d)^2, & if\ d \leq d_{min} \\ 0 \quad , & else \end{cases} \tag{3}$$

or

$$c_{danger}(c_i) = \begin{cases} (d_{min} - d)^2\ , & if\ d \leq d_{min} \\ (d_{opt} - d)^2 \quad , & else \end{cases} \tag{4}$$

With F the set of all frontier cells, $\chi_c^{c_g}$ the set of all possible paths from c to c_g, $l(C)$ the length of the path C, $c_{danger}(c_i)$ the cost function for the 'discomfort' of entering cell c_i, and α a weighting factor ≥ 0. d is the distance between cell c_i and the closest obstacle. To go to the target cell from any free cell, it is sufficient to follow the steepest gradient.

Equation (3) forces the robot to stay away from obstacles, no matter how far the robot gets away from them. This bears the risk that the sensors (with their limited range) cannot see any landmarks. This should be avoided, because the robot needs landmarks for localizing itself and for extending the map. Therefore, the cost function is extended in a way that the robot is encouraged to stay away from obstacles at a distance of d_{opt} and never gets closer than d_{min}, see equation (4).

Among the parameters that will influence how the environment is explored are the following:

- **inflate obstacles** (true / false)

  Inflate the detected obstacles with dimension of the robot in the path planning space to avoid collisions.

- **plan in the unknown** (true / false)

  Planned paths can go through unknown space.

- **wall following** (true / false)

  With wall following option set, the path planned for the robot will be less costly when at an optimal distance from obstacles.

- **security constant *α***

  α is a weighing factor ≥ 0 that determines how safe the path is. Close proximity to obstacles is punished with higher value of α. This gives the possibility to avoid narrow passages.

- **goal angle penalty**

The robot is encouraged to stay on its current path, to avoid unnecessary turning. A cost is added to frontiers which require the robot to turn; this cost is proportional to the angle difference. The closer the frontier to the robot, the harder it will be refrained from turning.

- **minimum obstacle distance**

  The minimum distance needed between the robot and obstacles to have a safe path. Paths planned for the robot will be less costly if they exceed this specified distance from the obstacles.

4 Results

4.1 Simulation Worlds

Simulation experiments are performed with the help of the 2D mobile robot simulator Stage [9]. Stage simulates a world as defined in a `.world` file. This file tells stage everything about the world, from obstacles, to robots and other objects. The simulation worlds used here are shown in figure 2.

4.2 Real Worlds

The real world experiments are performed in office corridors and labs with the Pioneer 3-AT robot, shown in figure 3. The Pioneer 3-AT is a 4 wheel drive robot, equipped with an embedded PC and 2D SICK LMS 200 Laser. The scanner measures distances in a plane up to 80m (cm accuracy) with an angular resolution of 0.5–1◦. The field of view is 180◦ resulting in 181–361 range measurements.

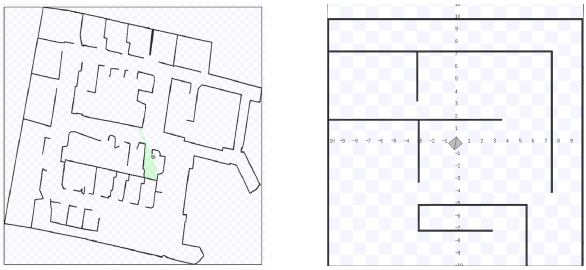

Fig. 2. Screenshots showing the simulation worlds in stage.

Fig. 3. Pioneer3-AT robot, equipped with 2D SICK LMS 200 laser.

4.3 Explore Package Results

In Figures 4 and 5 respectively the construction of the map during exploration with the *explore* package are illustrated by the image sequences obtained in simulation and the real world. The robot gradually explores the environment by planning paths to the generated goals and moving towards them while avoiding obstacles until exploration is finished.

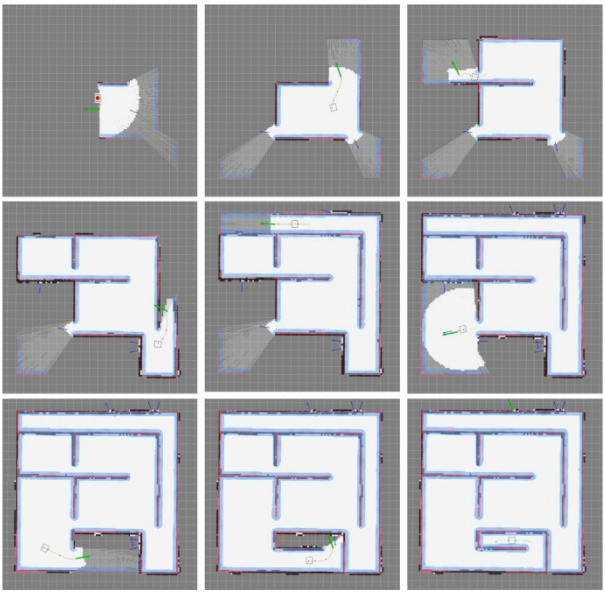

Fig. 4. Consecutive stages in frontier exploration mission in simulation.

The default local planner in ROS has been substituted by our own local planner which adheres to the *move_base* interfaces; as the currently available ROS planners are error prone. The *gmapping* package is used to produce the maps.

A tradeoff can be made between possible information gain of the frontier and the location and orientation of the frontier with respect to the robot's current position.

By varying the weights (1) of the different costs the robot can display different behavior while exploring the scene.

When two frontiers have more or less the same cost, it is possible that the robot will alternate between the two of them. By penalizing changes to the robot's current orientation, alternating between two frontiers can be reduced. Code has been added to take the distance between the robot and the frontier into account when adding extra costs due to orientation change. For frontiers in close proximity to the robot, the cost for turning is higher.

If the focus is on information gain, the robot explores the frontiers that gain the most information first. This is suited for an exploration mission that needs as much information as possible in a limited time frame. If the mission is to explore the whole environment, it is not needed to visit high information yielding frontiers first, as this will increase total exploration time and travel cost. Also, frontiers with high gain factor have no guarantee to produce the most information gain.

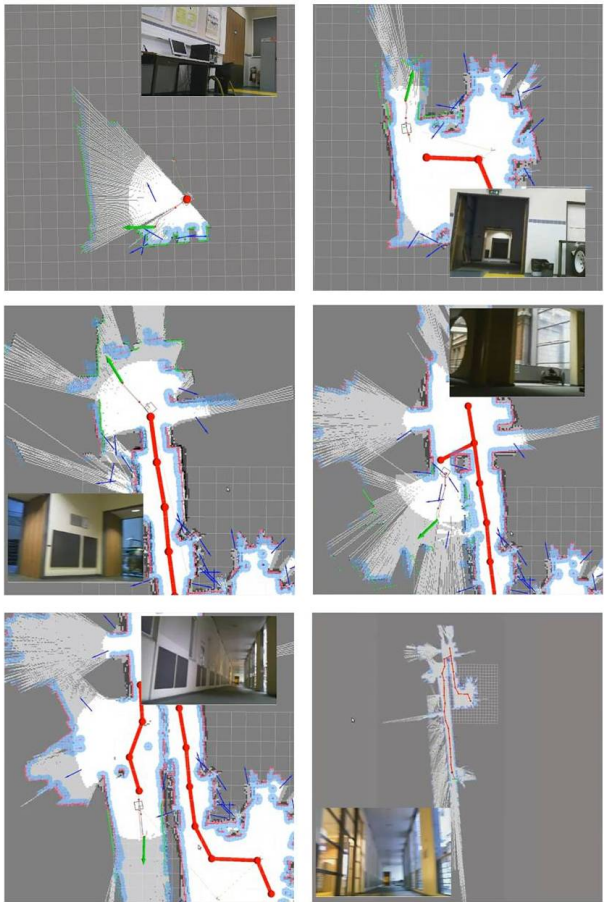

Fig. 5. Consecutive stages in frontier exploration mission in real world.

4.4 Hector_exploration_planner Package Results

The consecutive screen shots in figure 6 and 7 respectively show an exploration mission with the exploration transform in simulation and in real world. The exploration transform is visualized in a RGB color scheme. Red cells indicate a low cost distance to the next safe frontier; violet/blue cells indicate a high cost. The robot gradually explores the environment by planning paths to the generated goals and moving towards them while avoiding obstacles.

Instead of using the global planner in *move_base*, the trajectory is obtained from the steepest descent of the exploration transform, allowing several cost factors to come into play. The default local planner in ROS has been substituted by our own local planner which adheres to the *move_base* interfaces; as the currently available ROS planners are error prone. The *gmapping* package is used to produce the maps.

Figure 8 shows a comparison of an exploration mission where the wall following option is set and not set. If the wall following is enabled, the robot is encouraged to stay at optimal distance from the obstacles, as can be seen from costs in the figure.

When security constant α is set too small, the goal or the path to the goal can be too close to obstacles which can lead to potential collisions. Setting lower values for α will generally give shorter but more unsafe paths. Also narrow paths will no longer be avoided as the cost for passing through them is reduced. Setting high values for α will produce safe paths. Open spaces will be explored first, as the more dangerous narrow passages will be avoided due to high cost. Narrow passages are explored last. This can lead to longer exploration times. Although the behavior to avoid narrow passages can be beneficial in some scenarios.

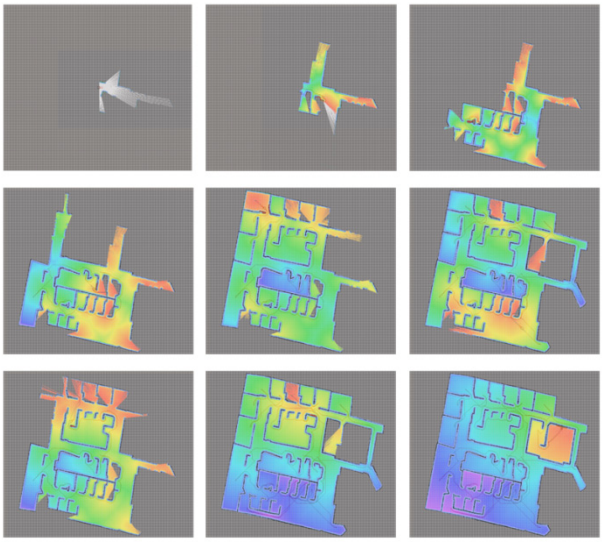

Fig. 6. Consecutive stages in frontier exploration mission in simulation. The exploration transform is visualized in a RGB color scheme. Red cells indicate a low cost to the next safe frontier, violet/blue cells indicate a high cost.

Setting the security constant α high will prevent the robot going through narrow spaces. But with high enough goal angle penalty, if the hallway is not too narrow it will finish the hallway before switching to other frontiers.

Figure 9 shows the traversed path of the robot with and without wall following for the particular simulation world in figure 6. The effect of the wall following option and the security distance on the exploration time is shown in figure 10. Exploration time increases with security distance. When the wall following option is set the exploration time is shorter for smaller security distances and longer for larger security distance. Of course a lot of other factors influence the results.

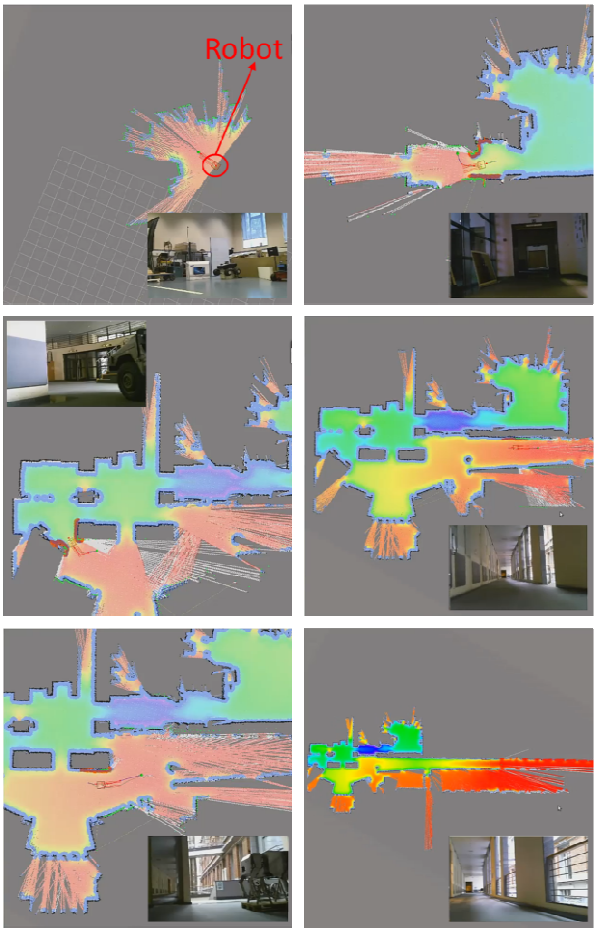

Fig. 7. Exploration of ground level building H of RMA. The exploration transform is visualized in a RGB color scheme. Red cells indicate a low cost to the next safe frontier, violet/blue cells indicate a high cost.

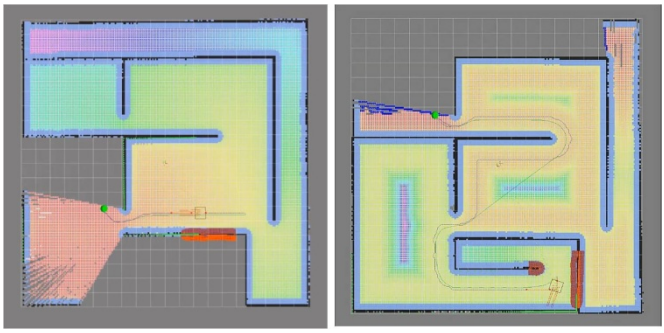

Fig. 8. Screenshots of an exploration mission where the robot is encouraged to stay at safe (left) and optimal (right) distances from the obstacles. The exploration transform is visualized in a RGB color scheme.

Red cells indicate a low cost to the next safe frontier, violet/blue cells indicate a high cost.

Fig. 9. Traversed path in simulation, with on the left for a safe distance of 0.6m, and on the right for wall following option with 0.6m.

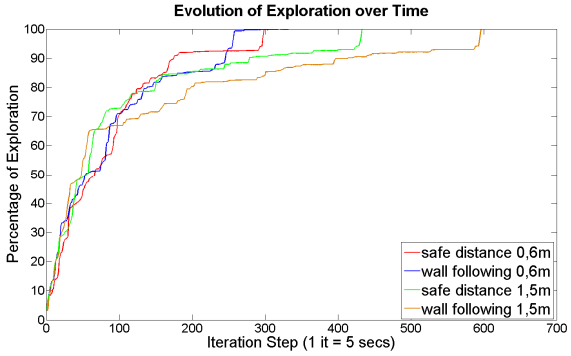

Fig. 10. Evolution of Exploration over time for different values of the security distance and wall following option for the specific case shown in figure 8.

5 Conclusion

In comparison with the *explore* package, the exploration transform method provides more options to manipulate the path the robot will follow by introducing extra costs with certain behavior. The used algorithms offer a reliable solution for autonomous exploration with a single mobile robot. They can cope with moving obstacles and re-plan when necessary. The next step is to let a multi-robot team explore and map an unknown environment. Each robot will produce partial maps; these maps of the individual robots will be merged in one global map. Based on this global map and the current positions of the robots in the team, it will be decided where each robot will go next to continue exploration.

References

1. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source robot operating system. In: Proc. IEEE Int. Conf. on Robotics and Automation (ICRA) - Workshop on Open Source Robotics, ROS (Robot Operating System), May 2009. http://www.ros.org/
2. http://wiki.ros.org/gmapping
3. Grisetti, G., Stachniss, C., Burgard, W.: Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In: Proc. of the IEEE International Conference on Robotics and Automation (ICRA) (2005)
4. http://wiki.ros.org/move_base, http://wiki.ros.org/navigation
5. http://wiki.ros.org/explore
6. Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., Klingauf, U., von Stryk, O.: Hector open source modules for autonomous mapping and navigation with rescue robots. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013. LNCS, vol. 8371, pp. 624–631. Springer, Heidelberg (2014)
7. Wirth, S., Pellenz, J.: Exploration transform: a stable exploring algorithm for robots in rescue environments In: IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR), pp. 1–5 (2007)
8. Yamauchi, B.: A frontier-based approach for autonomous exploration. In: Proc. IEEE Int. Symp. Computational Intelligence in Robotics and Automation (CIRA), July 1997
9. Gerkey, B., Vaughan, R.T., Howard, A.: The player/stage project: tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003), Coimbra, Portugal, pp. 317-323, June 2003. http://playerstage.sourceforge.net/