Autonomous Navigation and Exploration in a Rescue Environment

Daniele Calisi, Alessandro Farinelli, Luca Iocchi, Daniele Nardi
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
E-mail: `<lastname>@dis.uniroma1.it`

Abstract— **In this paper we present an approach to autonomous exploration of a rescue environment. Exploration is based on unexplored frontiers and navigation on a two-level approach to the robot motion problem. Our method makes use of a motion planner capable of negotiating with very fine representations of the environment, that is used to move in a cluttered scenario. A topological path-planner "guides" the lower level and reduces the search space.**

The two algorithms are derived from two widely-used probabilistic algorithms, currently successfully deployed in many robot applications, the Probabilistic RoadMap and the Rapid-exploring Random Trees; however, their adaptation to the rescue scenario requires significant extensions.

I. Introduction

Exploring the environment is one of the main tasks of a rescue robot. Exploration is needed in order to build the map and find victims. Our goal is to provide the robot with autonomous navigation capabilities, that ease operator's tasks and become necessary in presence of communication failures, thus increasing the effectiveness of navigation and victim search.

Exploration in a rescue scenario can be divided in two subtasks: *i*) decide where to go next, considering that the environment has to be explored as fast as possible and that there are places in which there are more chances to find victims; *ii*) move the robot to the target position, that involves the motion in a cluttered and rough-terrain environment.

The first decision involves various issues, depending on the kind of environment to be explored. In order to explore an unknown environment, a SLAM algorithm is needed, and in general it has to be coordinated with exploration choices (e.g. [10]). However, in small environments, like the ones that are considered in Rescue Arenas, the error accumulated in the localization or SLAM process is small enough and does not lead to serious difficulties when closing loops. Consequently, one can safely assume that exploration can be accomplished independently of SLAM (i.e. without specific requirements by the SLAM process, e.g. to re-visit well-known place in order to refine current localization hypothesis). In our system we use a scan-match based approach to localization ([2]), while the choice of which position to explore is based only on unexplored frontiers, as described in [12]. The choice of using a frontier-based exploration is due to the consideration that our focus is on unexplored areas of the map and the use of frontiers allows us to focus on unexplored areas, but also on the possibility of reaching them.

For what concerns the second decision, the solution of the general motion planning problem is computationally very expensive, even in the most basic formulations. The so-called "generalized mover's problem" has been proven to be PSPACE-hard [8]. This motivates the development of heuristic algorithms that are able to quickly find a solution in many cases of interest, though often relaxing some requirements such as completeness and optimality.

Moreover, in a rescue environment, some of the assumptions, made in order to solve this problem, do not hold. For example:

- one cannot assume that the robot moves at a safe distance from the obstacles and so the real shape and size of the robot have to be considered in order to negotiate narrow passages;
- the sensors are not accurate and suffer from a discretization error; moreover, some manoeuvre could unexpectedly fail, due to the presence of undetected obstacles, so one cannot assume to be able to do whatever manoeuvre;
- the map is not known a priori.

A straightforward application of path-planning and exploration techniques is therefore not successful in a rescue domain. Most notably, the major difficulties are caused by the need of a very accurate manoeuvring, but at the same time it is necessary to plan long paths, trying to avoid potentially difficult/blocked passages.

Recently, probabilistic methods have been introduced in robot motion in order to reduce the computational cost of the classical deterministic and geometric approaches. They are currently successfully used for solving many dofs manipulators and kinodynamic vehicles autonomous control. Those methods relax the completeness requirement to a probabilistic completeness, but provide computational tractability and formal guarantees on the behaviour of the algorithm. Two main probabilistic algorithms used in robot motion are the Probabilistic RoadMap [5] and Rapid-exploring Random Trees [6]; they both rely on the idea of computing a graph or a tree of feasible trajectories and then use one of them to reach the target.

We adopt a very common approach to robot navigation which makes use of a local planner, that uses as input the sensors readings and a local small map and can be more precise in steering the robot, and a global planner, that takes into consideration the whole knowledge of the environment. In this way, one can decouple the problem by first solving a simple path-planning problem and then find a trajectory that is able to follow that computed path.

In our approach we build a topological graph of the environment on which we can compute a topological path that subsequently the low-level motion planner will follow.

The main novelties of the proposed approach are the following:

- with respect to global path-planning, a new combination of Probabilistic RoadMap and Growing Neural Gas is proposed, featuring the ability to be used with a map that is built incrementally while exploring;
- with respect to the local motion planning, the Randomized Kinodynamic Planning approach has been extended with interleaved planning and execution, feedback control and on-line pruning.

In the rest of the paper we first introduce the local motion planner and then the topological high-level path planner, which computes the path that guide the execution of the local planner, in order to reduce the search space and speed up the computation.

II. The Motion Planner

In this section we first summarize the main features of the Randomized Kinodynamic Planner algorithm and then focus on our extensions, that make this method feasible for the use in a rescue environment.

A. Randomized Kinodynamic Planning

For the local motion planner we use an algorithm based on the Randomized Kinodynamic Planner (RKP) [7], which, in turn, is an extension of the well-known Rapid-exploring Random Tree (RRT) [6] that considers kinematic and dynamic constraints. These algorithms are probabilistic and build a tree which quickly and uniformly explore the search space. One of their strength is that motion constraints, such as robot kinetic and dynamic constraints, are easy to integrate into the algorithm. Moreover, the exploration of the search space is biased towards unexplored portion of the space, by random sampling points in that portions and "pulling" the tree towards those points. The features of this algorithm make it suitable for application in the rescue scenario, because it allows for planning movements, making it possible to know in advance if a trajectory could be followed (in contrast with reactive methods), and for fast (partial) replanning, that is necessary where the map is built incrementally and may change over time.

The Randomized Kinodynamic Planner starts by initializing a tree τ with the current robot position. A random pose x is then randomly generated (with some bias), and the node n in the tree, nearest to x is selected from the tree. A finite set of feasible motion commands C is then randomly generated and applied to the robot in the pose n, integrating over a finite time Δt. From the poses resulting from this process we choose the command $c \in C$ that moves the robot nearest to x, without colliding with an obstacle; then, we add a node in τ, which contains the resulting pose and the commands used to move the robot in that pose, linking this node with the one from which we started to integrate the motion command.

B. Extended RKP

The search space of the algorithm we use is that of poses (i.e. position and orientation) and velocities (speed and jog). We do not take into account dynamic constraints, due to the low velocities involved (caused by the fact that the environment is cluttered and partially unknown).

The use of an RRT-like approach allows for an easy specification of the set of constraints needed in order to navigate in the environment. Each constraint, indeed, needs only to be checked over the set C of motion commands. We can also indicate which manoeuvres are forbidden in a particular area of the environment (because, for example, they result in a stall, i.e. the robot is blocked by an undetected obstacle).

The main choices when using a RKP algorithm are:

- how to random generate the pose x;
- a distance function between poses;
- how to generate the set C of motion commands.

In order to speed up the tree building process, the generation of the pose is often somehow biased towards the target position. In a space without obstacles, the best solution is to use always the target position as input to the algorithm; random picking is necessary in order to create those manoeuvres that avoid obstacles and local minima. Since the use of the global path-planner reduces the possibility of local minima, we can increase the bias towards the goal, thus decreasing the time needed in the building process to reach the target. In addition, among the random commands generated, we can always insert the optimal command, i.e. the one that we would like to use in free space. In this way, if obstacles do not cause a collision from a certain point to the target, we quickly find the (optimal) trajectory.

We compute the distance between two poses using the following formula:

$$dist = \sqrt{(\Delta x)^2 + (\Delta y)^2 + k(\Delta \theta)^2}$$

that is a euclidean distance in the space of (x, y, θ) with a constant weighting factor k that measures how much the robot orientation is relevant (e.g. with a high k the robot first tries to turn in the right direction and then steers to reach the (x, y) position). For the generation of the set C of motion commands we join random motion commands with heuristic commands, e.g. those given by a control law in a space free from obstacles. We discard those commands that do not meet the initial constraints on motion commands.

The RRT and the RKP algorithms, assume to have the whole map and that it does not change over time; consequently they do not deal with the possibility to correct the plan once it has been computed and is being executed. In our case we have to deal with the imprecision and the discretization errors of the sensors and with the fact that the map is partially unknown; moreover, motion commands do not always lead to the desired behaviour because of the roughness of the terrain and because we cannot assume to be able to execute a motion command for exactly the amount of time needed. It is therefore necessary to consider a fast method to re-plan if the previous plan becomes invalid (i.e. it will bring the robot to a collision).

Furthermore, RRT and RKP usually build two trees, one from the current robot pose and one from the target pose, trying to connect them during each cycle, because this reduces the planning time. On the contrary, we build only one tree starting from the robot pose. In fact the target pose is usually near the unknown portion of the map and is likely to change a lot while the robot approaches that position. Moreover in our approach the target position for the low-level motion planner moves on a path generated by the high-level path-planner.

Summarizing, the main extensions to the RKP algorithm are the following:

1) Interleaved planning and execution: One of the main issues of our algorithm is that we interleave planning and execution of the plan. This has some advantage over the two-phase approach, as the plan is generated when (and where) we have information about the environment and we can build the plan while the robot is moving; the results are very small plans, that are indeed just parts of the global plan.

In this way we can use the algorithm to quickly move in the environment, without having to plan long trajectories, that in general will be made invalid during the subsequent exploration. While following the current plan step, we check the current error in robot pose and translate and rotate the tree accordingly, in order to keep the result of the tree (and hence the current plan) up to date.

2) Feedback control: Though the low speed used to explore the environment causes only a small error in trajectory following and the use of a good localization and mapping method minimizes the input errors, we believe that a closed-loop control is still needed to correct errors in the trajectory execution. For this reason we correct the motion commands using a feedback on the final foreseen position of the current plan. Another approach to feedback in RKP can be found in [3], that uses a single motion command (a feedback control law) at each iteration, making the method not suitable to easily set constraints on the plan.

3) On-line pruning: In narrow passages, the trajectory found on the tree becomes easily invalid, due to control and sensors errors, thus a re-planning phase is necessary. During this phase we can save a lot of computation if we partially maintain the tree previously computed. In [1] a different method is used, based on the computation of waypoints along the trajectory. Since our environment is not as dynamic as the one considered in that paper, we can keep a lot more information (i.e. the whole tree) and “prune” only branches that begin with a collision. Since the robot is moving, we can also prune the root and all branches that do not belong to the current tree of possibilities. In this way we have, at each cycle, only a limited set of nodes and branches, from which we can continue to grow the tree, in order to further explore the search space. The trajectory is computed on the current tree and this cannot lead to oscillations because at each node we only make a choice on which branch to follow; all other choices (branches) will then be pruned, thus limiting the size of the tree.

III. The Topological Path-Planner

In the following subsection we briefly describe the two algorithms that we successfully combined in order to obtain our global high-level path-planner, which is introduced in the next subsection.

A. The Probabilistic RoadMap and the Growing Neural Gas

Using complete algorithms to find the topology of the environment (e.g. Voronoi diagram) is very expensive and since we have a different map each cycle a probabilistic approach is more convenient also for the topological path-planner.

The most widely used probabilistic algorithm that builds a graph representing a roadmap of the environment is the Probabilistic RoadMap (PRM) [5]. The algorithm works by picking random positions in the configuration space and trying to connect them with a fast local planner. The problem with this algorithm is that it expects as input a map that does not change over time.

In order to overcome this limitation, we combine the PRM algorithm with Growing Neural Gas (GNG) [4]. GNG is a neural network with unsupervised learning, used to reduce the dimensionality of the input space. In this kind of network, nodes represent symbols and edges represent semantic connections between them; the Hebbian learning rule is used in many approaches to update nodes and create edges between them. Given a system which has a finite set of outputs, applying the Hebbian rule allows for modifying the network in order to strengthen the output in response to the input. Otherwise, given two outputs that are correlated to a given input, it is used to strengthen their correlation. For our concerns, the nodes (symbols) represent locations and the edges the possibility to go from one location to another. In this sense, we can use, together with the Hebbian learning rule, a simple visibility check in order to create a link between two nodes, as PRM does. GNG cannot be straightforwardly used in a robot motion problem, because the topological information is valid only when the graph has reached a state of equilibrium.

B. The Dynamic Probabilistic Topological Map

Our algorithm, that we call Dynamic Probabilistic Topological Map (DPTM), successfully combines PRM and GNG, taking into account the characteristics of the considered rescue environment. There are two main issues in this kind of algorithm: *1*) when to add a new node; *2*) when to add a new edge between two nodes.

First of all, in order to connect nodes we use the simple straight-line local planner to connect two nodes, usually used in PRM algorithms. We say that a node can “see” another if it can be connected through a straigh-line local path to the other. This relation is obviously symmetric.

1) Nodes: Since we want to have only those nodes that are needed to represent the topology of the environment, we do not add a new milestone each time a new position is presented to the network, but only if:

- the position cannot see any other node already in the network;
- the introduction of a new node makes it possible to connect two nodes already in the network.

If a node does not have to be added, we can use the Hebbian learning rule in order to reduce the error distortion in the set of positions represented by the node. In this way, we actually move the node in the center of place it represents (where the error distortion is minimum), thus incrementing the chance of connecting it with other nodes (as experimented in [11]). The second criterion is similar to that used in [9], but in our case each node could be a connection between two nodes, there is no fixed role.

2) Edges: For what concerns edges, we avoid to connect a node with all its neighbours, because they can be redundant for the topology representation of the environment. Instead, we use the Hebbian learning rule to connect the two nodes nearest to the current input position. We clearly add also the two edges that connect the new position to two nodes that cannot see each other.

The few nodes in the topological map make it easy to move them to different positions as topology modifies and check, at each cycle, if some edges are no longer valid, i.e. two nodes cannot see each other, and remove them.

C. Differences with PRM and GNG

In order to reach the same topological representation of the environment, the DPTM has a number of nodes that is in the order of 1% with respect to PRM. Moreover, with respect to GNG, the density of nodes is a function of the complexity of each portion of the map (how many nodes are needed to represent the topology), instead of being uniform, thus providing for a good trade between accuracy and relevance of the representation.

Moreover, using a DPTM we can extract the topology information of the environment, i.e. each path in the environment can be represented on the DPTM, while the PRM algorithm tries to achieve only the connectivity, eventually losing in the graph some connection existent in the environment. This means that with DPTM we can use some method in order to find the optimal path between two positions, while in general with PRM we cannot do it (we are not interested in other ways to reach the same target).

Another important difference is that the PRM algorithm needs to be stopped because it continues indefinitely to add nodes and edges to the graph, while the DPTM, given a static map, reaches a state in which the number of nodes remains almost constant.

Finally, DPTM easily adapts to the topological changes of the environment, making it useful in an environment whose map is incrementally build during exploration. The use of the the Hebbian rule moves the nodes in the center of the "places" they represent. Using a set of heuristics, we are able to remove some redundant or invalid nodes and edges. This process is done during the map building process in order to have at each cycle a network which turns to be very small and easy to maintain and to find topological paths on it.

Algorithm 1, shows the main cycle of DPTM algorithm, that can be run indefinitely. NEARESTNODE(N, q) returns a node $n_1 \in N$ which is the nearest to q and is visible from q. This function is called another time to get the second nearest node. If n_1 does not exists (no node can see the new position q), we add a new node to the graph in the position q. The function UPDATE(n, q) returns a node n^* resulting from the application of the Hebbian learning rule to the node n and the new position q, i.e. applying the following formula:

Algorithm 1: Dynamic Probabilistic Topological Map

```
BUILDDPTM($metricMap$)
(1)  $N \leftarrow \emptyset$
(2)  $E \leftarrow \emptyset$
(3)  while $true$
(4)      $q \leftarrow$ RANDOMPOSITION()
(5)      $n_1 \leftarrow$ NEARESTNODE$(N, q)$
(6)      $n_2 \leftarrow$ NEARESTNODE$(N \setminus n_1, q)$
(7)      if $\exists n_1$
(8)          $n_1^* \leftarrow$ UPDATE$(n_1, q)$
(9)          if $\neg$LOSEEDGES$(n_1, n_1^*)$
(10)             $n_1 \leftarrow n_1^*$
(11)         if $\exists n_2$
(12)             if VISIBILITY$(n_1, n_2) \wedge (n_1, n_2) \notin E$
(13)                 $E$.ADD$(n_1, n_2)$
(14)             else
(15)                 $N$.ADD$(q)$
(16)                 $E$.ADD$(n_1, q)$
(17)                 $E$.ADD$(n_2, q)$
(18)     else
(19)         $N$.ADDNODE$(q)$
```

$$
\begin{aligned}
n^*.x &= n.x + \eta(q.x - n.x) \\
n^*.y &= n.y + \eta(q.y - n.y)
\end{aligned}
$$

in which η is the learning factor, a constant parameter. In order to avoid to delete edges where a path between two nodes still exists, we first check if the node, in the new position, will lose some of its edges; this is done via the function LOSEEDGES(n, n^*): we update the node if and only if it does not lose any edge in the new position. The visibility check (using the function VISIBILITY(n_1, n_2)) ensures that there exists a straight path between two nodes, i.e. they can see each other. This function is used to check if the two nodes nearest to q can see each other: in this case, if they are not already connected, we make a link between them. Otherwise we check if we can connect the two nodes via the new position and eventually do it.

During each algorithm cycle, we also check the validity of all edges and eventually remove those that have become invalid.

IV. INTEGRATION

In this section we discuss how we integrate the high level path-planner and the low level motion planner algorithms in order to achieve a system able to steer the robot in a partially known and cluttered environment.

A. *The exploration module*

This is the module that points out the next target position to reach, i.e. the next frontier that has to be explored. This target is given as input to the topological path-planner, that tries to find a topological path from the current robot pose to the target.

B. *The "expanded" map*

In order to reduce the search space, we can compute the configuration space for a circular robot, whose radius is that of a circle inscribed in the real robot. This fast pre-computation will help to find the trajectories only along those topological paths that could be feasible for the robot. It is clear that if a path does not exists in this expanded map, it cannot exists in the real space (the robot is too large to pass through), while we cannot say anything if that path exists (that topological path may exists while no finite set of manoeuvres exists that can follow that path). This is the first step towards the reduction of the search space for the local motion planner.

C. *Choosing the target for the motion planner*

Once we compute a path in the graph (currently we use the Dijkstra algorithm, that is essentially a semplified version of the well known A* algorithm), we can try to follow it using the local motion planner. We choose the nodes along that topological path as target poses. We tried two methods: using the nodes as "waypoints" (as in [1]) or using the next topological node only if the plan reached the current one. As stated before, we do not need to reach exactly that pose, since topological path nodes only represent topological places, i.e. the Voronoi region of which they are generator. We experienced that the second method finds trajectory faster than the first, mainly due to the clutterness of the rescue environment.

D. *Motion planner (pseudo) failure*

The execution of the motion plan begins before the construction process ends, so it is possible that the planner cannot find the right trajectory that can steer the robot through that branch of the topological path. After a reasonable time, we request another topological path to the global path-planner, to avoid to remain trapped along a topological edge that cannot be traversed by the robot (though there exists a topological link between the two nodes). Due to the probabilistic nature of the algorithm, this cannot assure that a trajectory does not exist.

V. Results

In Figure 1 a partial map is shown and the corrispondent DPTM build on it. The thick line is the topological path, computed on the DPTM, that has to be followed by the motion planner. In Figure 2 we can see an example showing a difficult DPTM edge in a narrow passage, that the algorithm is able to deal with. In the first two images the trajectory found by the motion planner causes the robot first to move backwards, because it cannot turn or go forward; then, it find its way in the narrow passage. In the third and fourth image, while the trajectory is being followed,

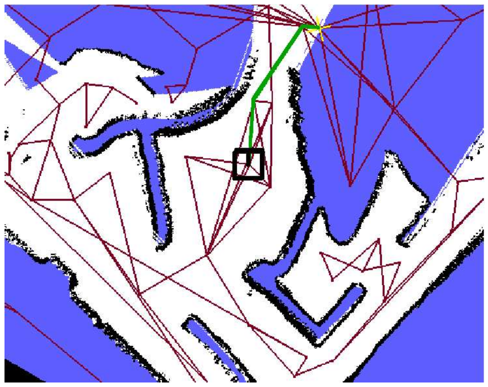
Fig. 1. The DPTM built on a partial explored map

the robot is shifted robot to the left, making the trajectory invalid and making it necessary to compute a new one. The whole process is done while the robot is running, it never stops to re-plan (unless the collision is found on the current step). The robot navigates at a speed of 10 cm/s and its size is 48x50 cm while the narrow passage is only 60 cm wide (the map is discretized at 50 pixels per meter).

In figure 3 we report the map built by our robot in the NIST yellow arena set up in Paderborn during the 2005 RoboCup rescue competitions. The gray line is the path done by the robot during the exploration. The robots successfully completed the exploration fully autonomously without ever colliding with any obstacles. As it is possible to see the arena is very cluttered, moreover the obstacles present in the arena are unstructured and difficult to identify.

VI. Conclusion

We presented an approach to autonomous exploration that is based on unexplored frontiers and two-level method for robot motion, based on two probabilistic methods that have been suitably extended in order to adapt to a rescue environment. The resulting method is able to steer a robot in a cluttered environment and has been tested both in simulation and in real rescue arenas. Thanks to the use of a RRT-like algorithm, the introduction of constraints on robot moving capabilities (e.g. minimum turning radius, forbidden manoeuvres in particular areas of the arena due to undetectable obstacles, ecc.), is straightforward.

Moreover, the use of a topological representation in the global level makes it simpler to interact with an operator or with a cognitive level.

We believe that during a rescue mission having autonomous navigation capabilities can be beneficial to the human operator. In fact, in such environments to correctly interpret robot sensor readings and to safely steer the robot, can be a hard and time consuming task for a human operator. On the other hand, using our approach the human operator can focus on more complex tasks (i.e. victim identification) taking control over the robot only when needed. Moreover, our system is robust to network failures. In fact, if a network breakdown happens, the robot can keep on safely exploring the arena until the connection is reestablished.

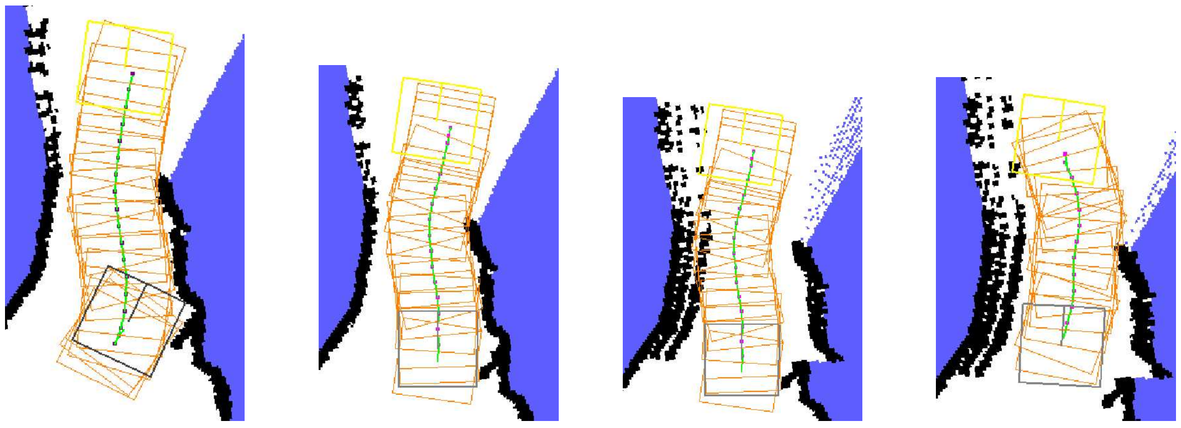
(a) 1 (b) 2 (c) 3 (d) 4

Fig. 2. An example showing the behaviour of the low-level motion planning algorithm

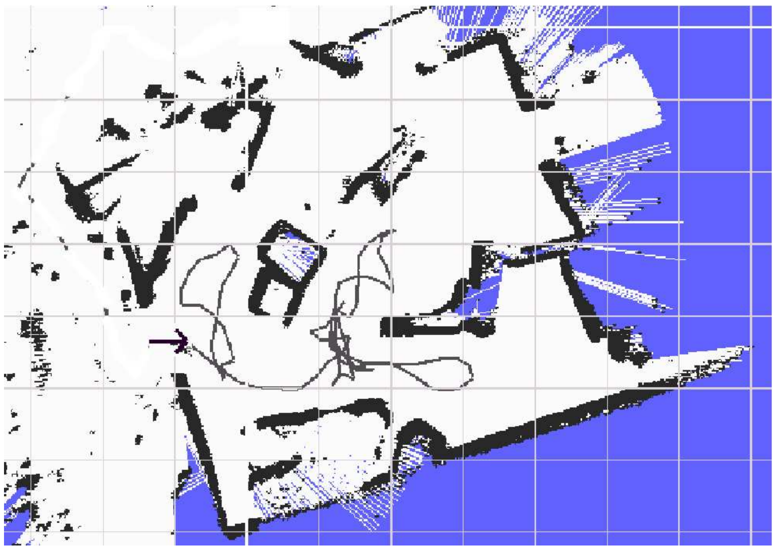
Fig. 3. This map has been built while autonomously exploring the arena in the RoboCupRescue Real Robots League of Paderborn RoboCup German Open competition

References

[1] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002, Switzerland, October 2002*, 2002.

[2] A. Censi, L. Iocchi, and G. Grisetti. Scan matching in the hough domain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'05)*, 2005.

[3] E. Feron E. Frazzoli, M.A. Dahleh. Real-time motion planning for agile autonomous vehicles. In *2000 AIAA Conf. on Guidance, Navigation and Control.*, 2000.

[4] Bernd Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995.

[5] L. Kavraki and J. Latombe. Probabilistic roadmaps for robot path planning. In *Practical Motion Planning in Robotics: Current Approaches and Future Challenges*, pages 33–53. K.G. and A.P. del Pobil, 1998.

[6] J. Kuffner and S. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'2000), San Francisco, CA, April 2000.*, 2000.

[7] S. LaValle and J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE International Conf. on Robotics and Automation*, pages 473–479, 1999.

[8] J. H. Reif. Complexity of the mover's problem and generalization. In *Proc. 20th IEEE Symp. on Foundations of Computer Sciences (FOCS)*, pages 421–427, 1979.

[9] T. Simeon, J. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. In *Proc. of IEEE IROS 1999*, 1999.

[10] C. Stachniss, G. Grisetti, and W. Burgard. Recovering particle diversity in a rao-blackwellized particle filter for slam after actively closing loops. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

[11] S. Wilmarth, N. Amato, and P. Stiller. Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. Technical report, Department of Computer Science, Texas A&M University, College Station, TX, nov. 1998.

[12] B. Yamauchi. A frontier based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation, Monterey, CA, July 10-11, 1997.*, 1997.