Robotics and Autonomous Systems 60 (2012) 628–638

Contents lists available at SciVerse ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Planning and obstacle avoidance in mobile robotics

Antonio Sgorbissa [*], Renato Zaccaria

DIST - University of Genova, Via Opera Pia 13, 16145 Genova, Italy

ARTICLE INFO

Article history:
Received 31 August 2011
Received in revised form
16 December 2011
Accepted 23 December 2011
Available online 8 January 2012

Keywords:
Mobile robots
Autonomous navigation
Planning
Reactive and local methods
Hybrid architectures

ABSTRACT

The paper focuses on the navigation subsystem of a mobile robot which operates in human environments to carry out different tasks, such as transporting waste in hospitals or escorting people in exhibitions. The paper describes a hybrid approach (Roaming Trails), which integrates a priori knowledge of the environment with local perceptions in order to carry out the assigned tasks efficiently and safely: that is, by guaranteeing that the robot can never be trapped in deadlocks even when operating within a partially unknown dynamic environment. The article includes a discussion about the properties of the approach, as well as experimental results recorded during real-world experiments.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

In the last twenty years, scientists have foreseen a close future in which Service Mobile Robots will be able to operate within human populated environments to carry out different tasks, such as surveillance [1,2], transportation of heavy objects [3–5], or escorting people in exhibitions and museums [6]. Unfortunately, and in spite of the huge number of works on the topic, very few existing systems are able to work continuously over a long period of time without performance degradation, i.e., the "six months between missteps" that Moravec advocated in 1999 [7]. It is commonly accepted that one of the causes of failure is that human populated environments cannot be completely known a priori since they are highly dynamic: this has a dramatic effect both on self-localization (i.e., it makes difficult to recognize and match features against environmental models) and on navigation (i.e., it is possible that a planned path is no more available or temporarily obstructed).

From a philosophical perspective, navigation in a dynamic environment has a straightforward solution: even if the path has been planned on the basis of the available a priori knowledge of the environment, the robot must also be free to take decisions in real-time on the basis of its current perceptions. However, how much should the robot be free to deviate from the planned path, is a question for which the answer is neither simple nor unique: in a metaphorical sense, it resembles the never-ending theological dispute between Free Will and Predestination.

From a technical perspective, Artificial Potential Fields (APFs) and similar force field-based models [8–10] have often represented a good solution to achieve a fast and reactive response to a dynamically changing environment; however, it has been widely demonstrated that they suffer from unavoidable drawbacks [11]. In particular, since the law of motion of the robot is basically determined by descending the gradient of the potential field generated by the goal and the obstacles, it is very likely for the robot to get trapped into a local minimum. The problem can be solved by invoking a planner to search for alternative paths; however, in presence of moving obstacles and sensor noise, significant deviations from the original path can lead to a deadlock configuration from which it is even harder to escape. This is further complicated by the fact that the robot is subject to geometric and kinematic constraints which puts severe limitations on its motion capabilities. As an extreme example, consider that the approach used by commercial Automated Guided Vehicles (AGVs) to avoid obstacles in real-world situations (e.g., in a corridor crowded with people) is often *not to avoid them at all*[1]: experience tells that, if the vehicle slows down or stops, warns people, and waits, it has more probability of success than trying to find a path to the goal by all means.

By recalling the dialectic between Free Will and Predestination, this article metaphorically proposes a solution (Roaming Trails) which is well exemplified by Roger Hestons epitaph in Edgar Lee Masters' Spoon River: "OH many times did Ernest Hyde and I / Argue about the freedom of the will. / My favorite metaphor was Picketts cow / Roped out to grass, and free you know as

[*] Corresponding author. Tel.: +39 010 3532801; fax: +39 010 3532154.
E-mail addresses: antonio.sgorbissa@unige.it (A. Sgorbissa), renato.zaccaria@unige.it (R. Zaccaria).

[1] See the description of a successful case study in www.swisslog.com/hcs-agv-memorialhermann.pdf.

0921-8890/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2011.12.009

far / As the length of the rope". Specifically, the paper describes a novel two-layered approach to mobile robot navigation that integrates two navigation methods: an off-line method, which generates a reference path on the basis of an a priori map, and an on-line method, which adapts the generated path to avoid static and moving obstacles, but only to a given extent (i.e., "as the length of the rope allows"). With respect to solutions proposed in the literature [12–20], the major contribution of the approach consists in the formalism adopted to represent a path: instead of a sequence of way points, Motor Schemata, or Elastic Strips, a path is represented as a Roaming Trail, i.e., a chain of diamond-shaped or elliptic areas whose boundaries define the maximum allowed distance for the robot to deviate from its path when generating the on-line trajectory. Roaming Trails, as shown in the following, are able by construction to prevent the robot from being trapped into deadlocks caused by static obstacles (i.e., obstacles depicted in the a priori map), as well as by moving and removable obstacles (i.e., obstacles not depicted in the a priori map). Smooth motion in presence of geometric and kinematic constraints is guaranteed by the control law adopted for navigation.

Section 2 describes Related work. Section 3 describes in details a prototypical two-level multi-agent navigation architecture, by focusing on the details of the agents involved. Section 4 describes how the same architecture can be modified to implement the Roaming Trail approach. Section 5 describes experimental results. Conclusions follow.

2. Related work

Some of the most successful approaches to obstacle avoidance [8,12] rely on the idea of computing artificial repulsive forces which are exerted on the robot by surrounding obstacles, and an attractive force exerted by the goal. All of these forces are then summed up to produce a resulting force vector that is used to control the motion of the robot. APF and similar approaches have well known problems [11], among which the presence of local minima in the field (e.g., in correspondence of narrow passages). In addition, if occupancy grids [21] are used to store sensor data, APF can produce oscillations in the resulting velocity vector (and hence on the robot path) due to the finite resolution of the grid. The problem is well described in [9], which argues that the latter two problems are both due to the fact that a huge amount of sensor data are summarized in just one force vector, and proposes the Vector Field Histogram (VFH) as a solution. In VFH, the concept of "resulting force" is substituted with the concepts of "valleys" towards which the robot is allowed to navigate. VFH has been applied both in indoor and outdoor robotic applications, and improvements are described in [22].

In spite of the many important differences, the approaches above share three drawbacks:

1. they are local methods, and therefore not able to deal with complex obstacles configurations encountered by the robot during navigation, e.g., narrow passages, cluttered areas, or *cul-de-sac* in which global planning capabilities or recovery strategies would be required;
2. they face obstacle avoidance assuming an ideal point-like robot, therefore requiring to transform the velocity vector into commands to actuators in a separate phase: this is not a trivial task whenever it is desirable to produce a smooth navigation trajectory in presence of geometric and kinematic constraints;
3. they do not put any upper bound on the maximum allowed deviation from the path while avoiding obstacles, thus possibly producing an undesirable behaviour in crowded environments, where it would be more efficient to simply stop, warn people, and wait.

To overcome problem 1, a class of approaches try to combine force field methods with traditional planning algorithms, by asking a symbolic planner to dynamically generate a sequence of local force fields which are free of minima (e.g., Motor Schemata [12] or Navigation Templates [13]), thus being able to guide the robot in different situations and areas of the environment while avoiding obstacles in real-time. Research in the field of the so-called "hybrid cognitive architectures", where the term "hybrid" refers to the concurrency of deliberative planning and reactive behaviours, has been very active up to one decade ago, producing notable examples such as AuRA, 3T, ATLANTIS [23,15]. Solutions based on the VHF method have been proposed as well. In [18] the enhanced method VFH* is presented, which verifies that a particular candidate direction guides the robot around an obstacle, by using the A* search algorithm and appropriate cost and heuristic functions. In [24] obstacle avoidance for an electric wheelchair is semi-automatically supported by the minimum vector field histogram (MVFH) method, whereas global planning is guaranteed by the manual intervention of the user. Obstacle avoidance in very dense, complex and cluttered scenarios has been considered in [25], and integrated with planning in [26]. The idea is that there are situations where it is more suitable to direct the motion towards a given zone of the space (that ameliorates the situation to reach the goal latter), rather than directly towards the goal itself. The algorithm finds sub-goals basing on the obstacle structure, and then associate a motion restriction to each obstacle to compute the most promising motion direction. The major limitation of the class of approaches above is that they require to perform planning in run-time, depending on the configuration of obstacles encountered during navigation. Moreover, they usually do not deal with problems 2 and 3.

Problem 2 is explicitly considered in another class of approaches. In [16] the Curvature–Velocity Method (CVM) is presented, that formulates obstacle avoidance as a problem of constrained optimization in velocity space. Physical limitations (velocities and accelerations) and the configuration of obstacles place constraints on the translational and rotational velocities of the robot. The robot chooses velocity commands that satisfy all the constraints and maximize an objective function that trades off speed, safety and goal-directedness. A variant of the approach is proposed in [27]. In a similar spirit, [17] presents the Dynamic Window approach (DW), which relies on the idea of performing a local search for admissible velocities which allow the robot to avoid obstacles while meeting kinematic constraints: in order to reduce computational complexity, the search is performed within a dynamic window which is centred around the current velocities of the robot in the velocity space, and only circular curvatures are considered. A theoretical treatment of convergence properties of the algorithm is proposed in [28]. Recently, [29] has proposed the Forbidden Velocity Map, a generalization of the Dynamic Window concept that considers obstacle and robot shape, velocity and dynamics, to deal with navigation in unpredictable and cluttered scenarios. CVM and DW are able to produce smooth trajectories while dealing with kinematic constraints; however, they still have the problem of local minima (a solution is proposed in [30] by introducing a planning stage in DW). Obstacle avoidance is solved as a non-linear feedback control problem in [20]: path following is achieved by controlling explicitly the rate of progression of a "virtual target" to be tracked along the path [31,32], and obstacle avoidance relies on the deformable virtual zone principle, that defines a safety zone around the vehicle, in which the presence of an obstacle drives the vehicle reaction. However, as stated by authors, the combination of path following with a reactive obstacle avoidance strategy has a natural limitation coming from the situation where both controllers yield antagonist system reactions. This situation leads to a local minimum, where a heuristic switch between controllers is necessary. Again, problem 3 is not considered.

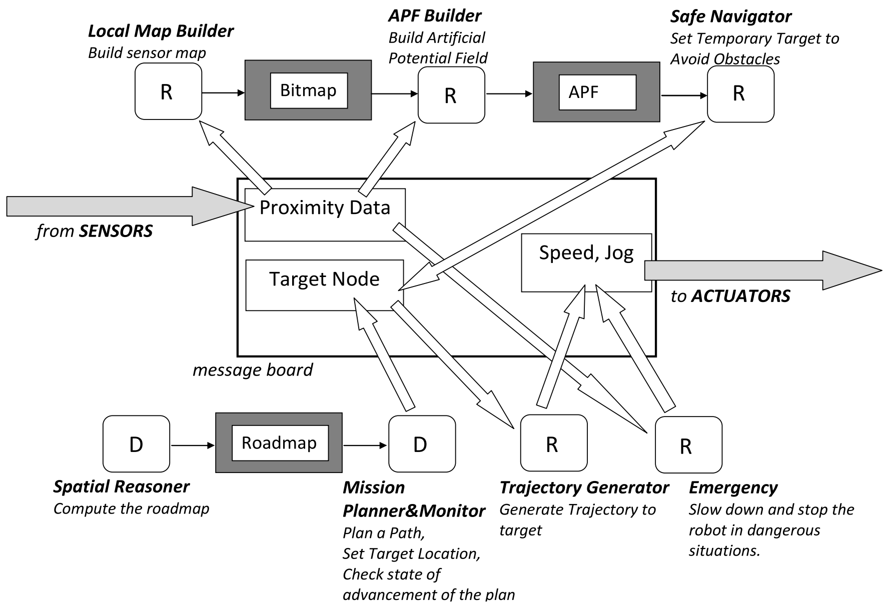

Fig. 1. Multi-agent navigation system.

Problems 1–3, are faced in an integrated fashion in [14,19], where a path is planned off-line, and then the whole path is ideally deformed in real-time on the basis of forces exerted by surrounding obstacles. In particular, the initial path is augmented by a set of paths homotopic to it, represented implicitly by a volume of free space in the work space which describes the maximum allowed deviation from the path. During execution, reactive control algorithms are used to select a valid path from the set of homotopic paths, using proximity to the environment in a sense very similar to [8]. The concept of path deformation is considered also in [33], in which the current path is described as a mapping from an interval of real numbers into the configuration space of the robot, and iteratively deformed in order to get away from obstacles and satisfy the nonholonomic constraints of the robot. The approach has been shown to work with complex non-holomic systems (e.g., a trailer) with complex shapes. Approaches based on path deformation avoid local minima since the path connection is preserved during deformation, and they put boundaries on the maximum allowed deviation from the original path. However, they have higher computational and memory requirements than the approach proposed in this article, because they require either to memorize a set of alternative paths, or to compute the path deformation in run-time.

3. Navigation architecture

A prototypical two-level navigation architecture is introduced: many different choices are possible in the design process and, consequently, many different models have been presented in the literature. However, it seems that the following considerations are valid for most existing approaches. The described architecture has been implemented on real robots whose behaviour has been evaluated through experiments, thus providing a starting point to incrementally implement the Roaming Trail approach above it. In the following, we ignore all the problems related to position uncertainty, by assuming that a sufficiently accurate self-localization subsystem is available, such as the laser-based localization system described in [34].

As usual, both a deliberative and a reactive component are present in the architecture, each component being implemented as a multi-agent system exploiting the ETHNOS real-time programming environment [35]: only agents related to navigation are shown in Fig. 1. Agents (rounded squares) communicate by posting messages (rectangles) on a distributed message board *or* by accessing shared representations (shaded rectangles). *Reactive agents* (labelled with an "R") are responsible of trajectory generation and other simple reactive behaviours. *Deliberative agents* (labelled with a "D") are responsible of generating sequences of actions in order to accomplish a given missions and monitor the state of advancement of the plan.

3.1. Deliberative agents

Spatial Reasoner has the purpose of building a graph-like representation of the environment referred to as the roadmap $\mathcal{R}$: the nodes of $\mathcal{R}$ represent significant locations in the environment (i.e., the centre of a room, a T-junction, a door, or the middle of a corridor), whereas links store information about the connectivity of these places. The roadmap is built starting from an a priori geometrical map $\mathcal{M} = \{s_i\}$, $i = 1 \ldots S$, which represents the workspace, where s_i is an oriented segment defined by an ordered pair of points. Ordered segments allow for defining not only the boundary between obstacles and the free space, but also the subspace which must be interpreted as free: for instance, if a segment goes from A to B, one can assume that the free space lies on the right half plane. Obstacles are represented through closed polygons, and the free space is assumed to be finite, i.e., it exists a closed polygon made with segments of $\mathcal{M}$ such that no free space exists outside the polygon.

The approach adopted to extract information about the topology of the free space relies on a popular method based on the Voronoi diagram [36], described in Algorithm 1 (see, among the others, an alternative approach proposed in [37]).

Algorithm 1 is self-explaining. It is worth spending a few additional words about step 1, which requires to solve a typical problem of computational geometry, i.e., checking if a cell $m_i \in \mathcal{G}$ lies inside or outside a polygon defining the boundary between obstacles and the free space. Since all polygons are closed and made of oriented segments, it can be solved by ideally tracing an arbitrary line passing through the centre of m_i, and by finding the closest segment s_j which intersects the line (if the line does not intersect any segment, m_i does not lie in the free space, since the latter is assumed to be finite). Then, it is possible to classify m_i as *free space* or *not free space* depending on where the centre of m_i is located with respect to s_j.

Algorithm 1 Compute Roadmap $\mathcal{R}$

```
Require: $\mathcal{M}$
Ensure: $\mathcal{G}$, $\mathcal{R}$
1: The workspace is sampled with arbitrary resolution, producing
   a grid $\mathcal{G} = \{m_i\}$, $i = 1 \dots M$, where each cell $m_i$ is either
   labelled as *free space* or *not free space*, using the information
   in $\mathcal{M}$;
2: each cell of the grid $m_i$ labelled as *free space* produces a Voronoi
   node if and only if its centre is (approximately) equidistant
   from the three closest segments $s_j, s_k, s_l \in \mathcal{M}$;
3: the resulting nodes $n_i$, $i = 1 \dots N$, are added to the roadmap $\mathcal{R}$
   and labelled with the position of the corresponding cell $m_j$;
4: for every couple of nodes $n_i$, $n_j$, $i \neq j$, an undirected link is
   established in $\mathcal{R}$ if and only if it is possible to draw a straight
   line which connects their corresponding positions without
   intersecting any segment in $\mathcal{M}$.
```

Mission Planner & Monitor is a goal-based agent responsible of plan selection and adaptation, allowing the robot to plan and execute high level tasks such as "go to office A", "call elevator" or "transport waste". These tasks, depending on the current robot context, may be decomposed into many, possibly concurrent, subtasks such as "localise, go to the door, open door" and finally into primitive actions such as "go to configuration (x_i, y_i, θ_i). In the following, we focus on navigation tasks: when *Mission Planner & Monitor* receives a mission (i.e., a location in the environment to be reached) it executes Algorithm 2.

Algorithm 2 Compute Path $\mathcal{P}$

```
Require: $\mathcal{M}$, $\mathcal{R}$
Ensure: $\mathcal{P}$
1: A couple of nodes $n_s$ and $n_g$, corresponding to the start and goal
   positions, are added to $\mathcal{R}$;
2: for every node $n_i \in \mathcal{R}$, an undirected link is established be-
   tween $n_i$ and $n_s$ – respectively, $n_g$ – if and only if it is possible to
   draw a straight line which connects the positions of $n_i$ and $n_s$ –
   respectively, $n_g$ – without intersecting any segment in $\mathcal{M}$;
3: the resulting roadmap is searched for the shortest path
   between $n_s$ and $n_g$ using the A* algorithm, thus producing
   a sequence of nodes $\mathcal{P} = (n_1 \dots n_t)$ in $\mathcal{R}$ to be visited in
   sequence, where $n_1 = n_s$ is the first node (corresponding to
   the robot current position) and $n_t = n_g$ is the final destination.
```

At any time, *Mission Planner & Monitor* knows the location of the next node n_i to be reached, it posts this information to the message board for *Trajectory Generator* and the other reactive agents which are responsible for the robot motion, and periodically checks the state of advancement of the plan.

3.2. Reactive agents

From the point of view of reactive agents, the problem to be solved is point-to-point navigation in an unknown environment. In fact the a priori map $\mathcal{M}$ is used exclusively to build the roadmap $\mathcal{R}$ and to consequently choose the nodes $\mathcal{N}$ to be reached in sequence: since we want the robot to deal with a dynamic environment where unpredictable things can happen, the robot must be able to find a way to the current target node n_i by relying on its local perceptions. The agents involved in the process are the following.

Trajectory Generator retrieves from the message board the last posted node n_i, i.e., the next location to be visited in the environment. Then, it is capable of generating and executing smooth trajectories from the robot current configuration $q = (x_r, y_r, \theta_r)$ to a target configuration (x_i, y_i, θ_i) relying on a closed-loop control function, where the target orientation θ_i can be manually specified or automatically computed, e.g., to minimize the path's curvature. In the current implementation, the control function is a biologically inspired motion generator called ξ-model [38]. As already stated, the smoothness of the trajectory is a fundamental characteristic, whenever we ask robots to operate in the real world: mobile platforms for real world applications (such as transportation of heavy loads) have severe geometric and kinematic constraints, to reduce both the stress on their mechanical parts and errors in dead reckoning (e.g., non-holomic or quasi-holonomic geometries are preferred to holomic ones). To deal with these constraints, *Trajectory Generator* is able to generate and execute a smooth trajectory in closed-loop, by producing a sequence of speed and jog values (i.e., linear and angular velocity) which are transformed into velocity commands to be issued to the rear motors of a differentially driven vehicle. However, *Trajectory Generator* cannot deal with obstacles in the robot path.

Local Map Builder, *APF Builder* and *Safe Navigator* are the agents responsible of obstacle avoidance. In Fig. 1 two shared representations (indicated as shaded rectangles) can be observed: the bitmap, an ecocentrical statistical dynamic description of the environment which is periodically updated through sensor readings [21,39], and the APF, based on the bitmap and on direct sensor information [10]. *Map Builder* and *APF Builder* update these representations to maintain consistency with the real world on the basis of sensor data. *Safe Navigator* periodically executes a virtual navigation in the APF (continuously aligned with the real world), thus determining a segment of safe trajectory which allows the robot to successfully avoid obstacles while heading to the goal. To achieve this, *Safe Navigator* reads the target node n_i produced by *Mission Planner & Monitor* and uses it to compute a new target node n'_i as the endpoint of the generated safe trajectory: n'_i is then posted to the message board and becomes available to *Trajectory Generator* for smooth obstacle avoidance. Fig. 2 should help to clarify this concept: when in position q', *Mission Planner & Monitor* sets n_1 as the current node; *Safe Navigator* performs a virtual navigation in the APF and moves the current target to n'_1; *Trajectory Generator* computes a smooth trajectory to n'_1 and the robot starts moving towards it. However, n'_1 is never reached by the robot, since at the next time step (when the robot is in q'') *Safe Navigator* computes a new target n''_1 which is fed to *Trajectory Generator*: this procedure is iterated until the real target node n_1 is eventually reached. The approach guarantees, at the same time, smoothness (as a consequence of the control law implemented by *Trajectory Generator*) and safety (since *Safe Navigator* reactively compute a free path in the APF), and share some similarities with other virtual target approaches [31,32,20].

Emergency Handler intervenes whenever an unforeseen collision is imminent. Since the actual trajectory followed by the robot differs from the virtual trajectories periodically generated in the APF, it is possible for the robot to collide with some obstacles in the environment. *Emergency Handler* prevents this from happening: on the basis of raw proximity data, in critical conditions it suddenly switches off the smooth navigation mechanism, and slows down or even stops the motors. Next, it forces the robot to turn in place until a free passage is found, before switching to the normal navigation mode again. This allows the robot to correctly and safely avoid obstacles even when the environment is highly dynamic, and a person or another robot is quickly approaching without allowing the robot to manoeuvre around it. Clearly, the resulting trajectory will no longer be smooth.

In spite of its good properties, the navigation system described in this Section has the typical drawbacks of systems relying on local perceptions and navigation strategies (see Section 2). First, since obstacle avoidance ultimately relies on APF, we have to deal with local minima in the potential field, which can prevent the robot from reaching the target node if an appropriate escape strategy

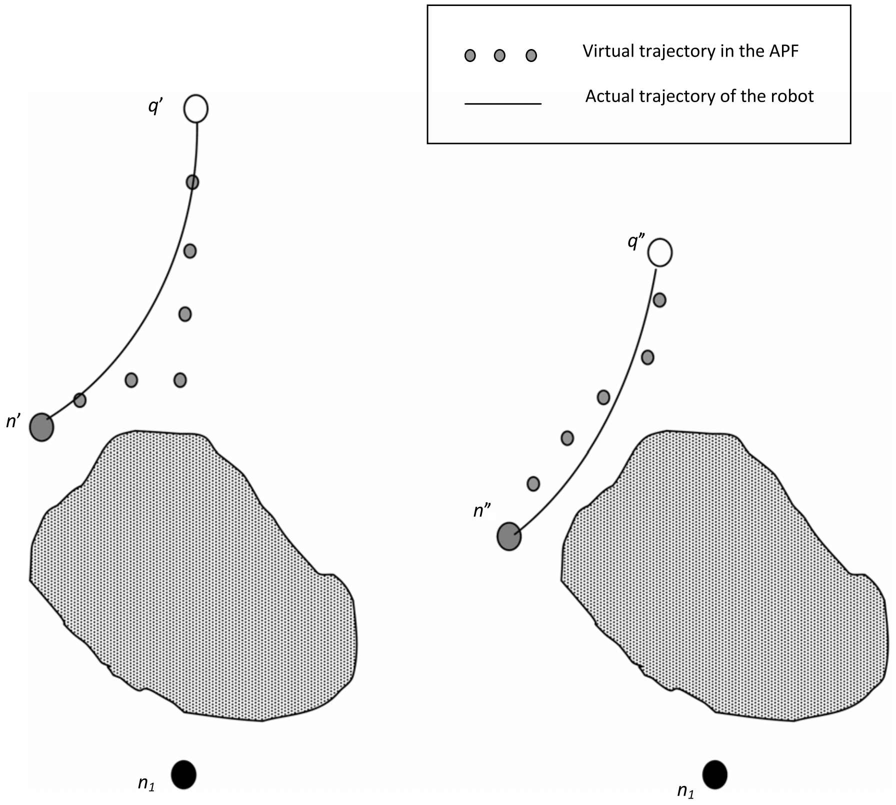

Fig. 2. Virtual navigation and real trajectory.

has not been implemented. Second, the navigation system does not provide a policy to determine how much the robot should be allowed to deviate from the original path, and an unlucky configuration of obstacles (or a nasty person) could cause the robot to move very far from its original path in the attempt to search for an alternative solution. Roaming Trails offer a solution to deal with these problems in an integrated fashion.

4. Roaming trails

To begin with, obstacles in human environments are categorized into three classes, defined as follows.

1. *Static objects*: this class includes walls and furniture that are guaranteed to stay still during the robot mission, and therefore can be depicted in the a priori map $\mathcal{M}$.
2. *Moving objects*: this class includes people, as well as objects carried around by people (e.g., suitcases and bags) that can be temporarily abandoned and lie on the robot path, but are not depicted in the a priori map $\mathcal{M}$.
3. *Removable objects*: this class includes objects (usually furniture, e.g., chairs) that are static in most cases, but can be occasionally displaced and lie on the robot path. As well as moving objects, removable objects are not depicted in the a priori map.

The Roaming Trail approach is able to deal very efficiently with environments in which most obstacles encountered by the robot during motion are of type 1 and 2. In addition, it is able to manage situations in which the robot encounters an object of type 3, even if less efficiently.

To achieve this, the underlying idea is the following: instead of planning the path on the basis of the topology of the roadmap $\mathcal{R}$, a more complex representation is extracted from the a priori map $\mathcal{M}$ and used for path planning and navigation. An example is shown in Fig. 3: some of the roadmap nodes are connected by means of diamond-shaped areas $\mathcal{RT}_{ij}$, which – taken together – constitute a Roaming Trail leading from the start to the goal. Specifically,

1. each area $\mathcal{RT}_{ij}$ is defined by the triplet (n_i, n_j, σ), where n_i, n_j are two roadmap nodes, and σ is an additional parameter which determines the dimensions of the area;
2. Roaming Trails are drawn in such a way that the intersection of each area $\mathcal{RT}_{ij}$ with the free space in $\mathcal{M}$ is always a convex area.

During navigation the robot is allowed to deviate from its path to avoid obstacles on the basis of reactive navigation strategies, but it is never allowed to exit from the area $\mathcal{RT}_{ij}$ which connects the current target node n_j with the previous target node n_i. Thus, since the robot is constrained to move within a convex area which includes the location of the target node, in presence of static obstacles it is guaranteed to reach the target by following a straight line. If we consider also moving obstacles (whose location is obviously not shown in the a priori map), the robot has two choices: avoiding them or simply stopping and waiting for them to move on. In both case, it can happen that the robot is temporarily unable to find a way to the goal, or even trapped in a deadlock caused by moving obstacles. However, if humans are collaborative as they are expected to be, they will let the robot pass: more important, it cannot happen that the robot, while trying to avoid a moving obstacle, ends its path in a concavity formed by an unlucky configuration of static obstacles (e.g., a *cul-de-sac* formed by the left wall and the two tables in Fig. 3): in fact, the robot is stopped as soon as it reaches the boundaries of the Roaming Trail. One could argue that the system relies on the fact that humans are collaborative and, sooner or later, let the robot pass. However, this is a limitation of every system programmed not to hit people to meet safety requirement: a single non-collaborative human can stop the robot for an arbitrary time, if the former is faster than the latter, whichever algorithm is adopted for navigation.

The presence of removable objects poses major limitations: in fact, even when the robot remains within the boundaries of Roaming Trails, it is always possible that it gets trapped into a concavity formed by removable objects (e.g., chairs or waste baskets) whose position is different from time to time, and hence

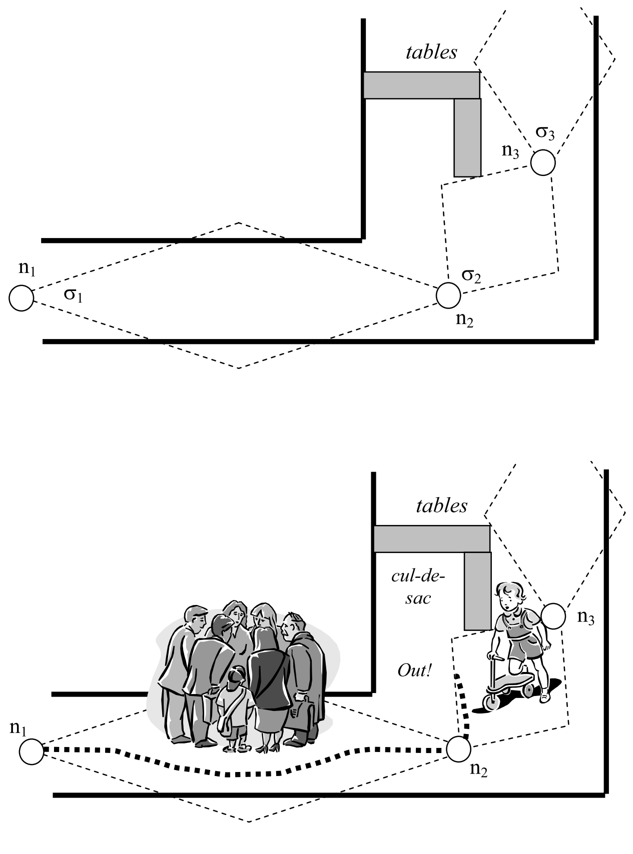

Fig. 3. Top: a priori map and Roaming Trails. Bottom: trajectory of the robot (dotted line) in presence of people.

are not depicted in the a priori map. Obviously, waiting for a chair to move on reveals not to be a winning strategy, unless there is a human in the neighbourhoods that quickly understands the situation and decides to directly intervene to help the robot! One could argue that, in this case, it would be more efficient to let the robot wander outside Roaming Trails, instead of waiting for the intervention of a *deus ex machina*. Of course, after some times that the robot is still, this can be done. However, we still claim that – in many realistic situation – Roaming Trails reveal to be a more efficient and safe strategy to merge a priori knowledge and planning with sensor-based reactive methods.

It should also be noticed that diamonds are not necessarily the best shape. In fact, this choice is rather arbitrary, and different shapes could be considered as well, e.g., ellipses, rectangles, or generalized cones. This means also that known methods in the literature for motion planning could be adapted to work in a similar manner (e.g., methods for exact or approximate cell decomposition [36]). We now show in details how the prototypical architecture introduced in the previous Section can be adapted to implement these concepts.

4.1. Deliberative agents

Spatial Reasoner behaves as described in Algorithm 1, by building the roadmap $\mathcal{R}$ in four steps. However, step 4 is now modified to meet the constraints put by Roaming Trails, i.e.,

4. for every couple of nodes n_i, n_j, $i \neq j$, an undirected edge is established in $\mathcal{R}$ if and only if the intersection between the area $\mathcal{RT}_{ij} = (n_i, n_j, \sigma)$ and the free space in $\mathcal{M}$ is a convex area.

An analogous modification is made to step 2 of Algorithm 2, required by *Mission Planner & Monitor* to connect the start and the goal node to $\mathcal{R}$. Moreover, instead of searching for the shortest path, step 3 of Algorithm 2 can be modified to search for the path with the lowest number of nodes: less nodes obviously corresponds to less and bigger areas $\mathcal{RT}_{ij}$, which give the robot more freedom to deviate from the nominal path to avoid moving and removable obstacles. As in the previous case, *Mission Planner & Monitor* communicates the location of the next node to be reached to *Trajectory Generator* and to the other reactive agents which are responsible for the robot motion, and periodically checks the current state of advancement of the plan.

4.2. Reactive agents

The reactive agents *Trajectory Generator*, *Local Map Builder*, *APF Builder* and *Safe Navigator* behave as in Section 3.2. On the opposite, *Emergency Handler* is different, since it now implements a set of rules to inhibit the behaviour of other agents not only when a collision is imminent, but also when the robot is heading towards an area of the workspace in which it could be trapped in a deadlock. Specifically, *Emergency Handler* intervenes if

1. the robot, when travelling from n_i to n_j, reaches the boundaries of an area $\mathcal{RT}_{ij} = (n_i, n_j, \sigma)$, which can happen when the robot is trying to avoid a moving or a removable obstacle;
2. the robot gets very close to an obstacle, which can happen either because the robot is trapped in a local minimum or because it has not enough space to manoeuvre around it.

If either condition verifies, *Emergency Handler* slows down and stops the robot, and turns it towards the next node n_j. Next, if the path is free, it allows the robot to move again; otherwise, it forces the robot to wait until all the moving obstacles have moved on.[2] Remember that, by construction, n_j is reachable from every position within $\mathcal{RT}_{ij} = (n_i, n_j, \sigma)$, since the robot motion is always constrained within a convex area. As a consequence, if an obstacle intersects the straight line connecting the robot position and the target, it necessarily is a moving or a removable obstacle which is not depicted in the a priori map.

5. Experimental results

Figs. 4–9 show how Roaming Trails are built. First, the workspace is sampled at an arbitrary resolution R, obtaining a grid $\mathcal{G}$ with M cells (Algorithm 1, step 1). Next, every cell m_i which is labelled as *free space* is searched for a Voronoi node, producing $N \ll M$ nodes (steps 2 and 3). Notice that, depending on the threshold chosen when checking if m_i is equidistant from the three closest segments, it can happen that a number of neighbouring nodes is found. To avoid this, experiments are performed with the additional constraint that, whenever a Voronoi node is found in a cell m_i, the algorithm does not search for additional nodes within a square of side P (arbitrarily chosen) centred in m_i: as a consequence, the minimum distance between nodes cannot be lower than $(P-1)/2$. Finally, couples of nodes n_i, n_j are connected in $\mathcal{R}$ whenever it is possible to draw an area $\mathcal{RT}_{ij} = (n_i, n_j, \sigma)$ whose intersection with the free space is convex (step 4). When the planner is asked to plan a path between a start S and a goal G, it updates the roadmap by adding the start and goal nodes (Algorithm 2, steps 1 and 2), and computes a chain of areas $\mathcal{RT}_{ij}$ with proper characteristics (step 3).

It is worth spending a few words about time complexity. Step 1 of Algorithm 1 has $O(SM)$ complexity: for every cell of the grid $m_i \in \mathcal{G}$, $i = 1 \dots M$, it is necessary to consider all segments of the map $s_j \in \mathcal{M}$, $j = 1 \dots S$, to check whether the cell lies in the

[2] Optionally, if the robot waits for too long, rule 1 can be disabled, thus letting the robot free to wander at its own risk.

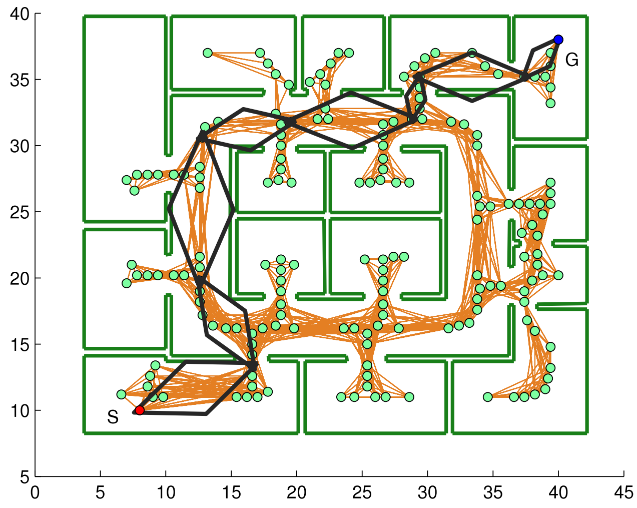

Fig. 4. Diamond shapes, $R = 5$ cells per metre, $P = 9$, solution with the least number of links.

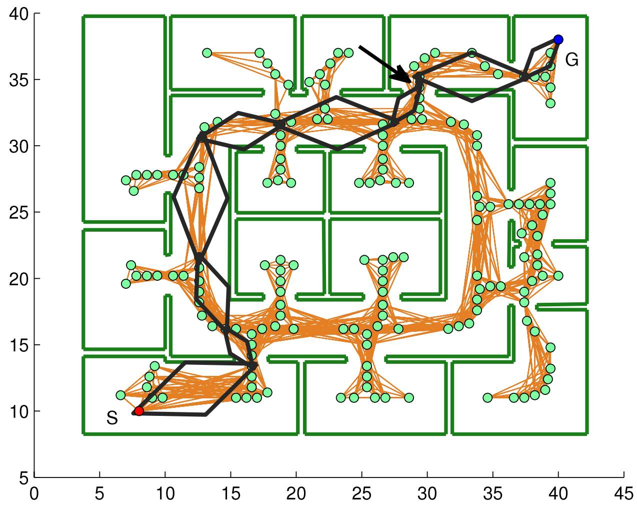

Fig. 5. Diamond shapes, $R = 5$ cells per metre, $P = 9$, shortest path solution.

free space or not. Finding Voronoi nodes in steps 2, 3 has $O(SM)$ complexity: there are at most M cells in the grid which are labelled as *free space*, and for every cell m_i it is necessary to compute the three closest segments $s_j, s_k, s_l \in \mathcal{M}$. Adding links in step 4, given that N nodes have been produced in the previous steps, has $O(SN^2)$ complexity: for every couple of nodes $n_i, n_j, i \neq j$, it is necessary to check whether the intersection between $\mathcal{RT}_{ij} = (n_i, n_j, \sigma)$ and the free space in $\mathcal{M}$ is convex, which – once again – requires to consider each segment $s_l \in \mathcal{M}, l = 1 \dots S$. Updating the roadmap in step 2 of Algorithm 1 has $O(SN)$ complexity, since it is necessary to check whether the start and the goal can be linked or not with any of the pre-existing N nodes, whereas the complexity of planning in step 3 is the complexity of the A* algorithm, which – in the worst case – is exponential in the length of the solution, and is known to be optimum among search algorithms. In general, the roadmap in Algorithm 1 can be computed off-line, since it does not depend on the particular navigation problem to be solved. On the opposite, a new path must be computed whenever the robot is assigned a new mission, and therefore the computational complexity of Algorithm 2 is an issue which deserves attention.

Experiments have been performed on a 2.66 GHz Intel(R) Xeon(R) CPU.[3] The problem in Fig. 4 has been solved by setting $R = 5$ cells per metre and $P = 9$, and by asking the planner to find the solution with the least number of links (which turns out to be 8). The time required to build $\mathcal{G}$ by classifying each of the M cells as *free space* or *not free space* is $t(\mathcal{G}) \approx 1.58$ s; the time required to build $\mathcal{R}$ is $t(\mathcal{R}) \approx 1.61$ s; the time required to update $\mathcal{R}$ and plan a path $\mathcal{P}$ is $t(\mathcal{P}) \approx 0.02$ s. The solution in Fig. 5 has been found by asking the planner to search for the shortest path, instead of the path with the least number of links. The resulting path consists of 10 links, corresponding to smaller areas $\mathcal{RT}_{ij}$ than the previous case: notice, for instance, the very small area which connects the 8th and the 9th nodes in the path (pointed by arrow). The solution in Fig. 6 uses a different shape for Roaming Trails, i.e., ellipses instead of diamonds: the robot is allowed to deviate more from the planned path, especially in the proximity of nodes, but a bigger number of links is required. The time required to find a solution is approximately the same in all cases.

The complexity analysis tells that, if required to reduce computational cost, (a) the resolution R of the grid can be

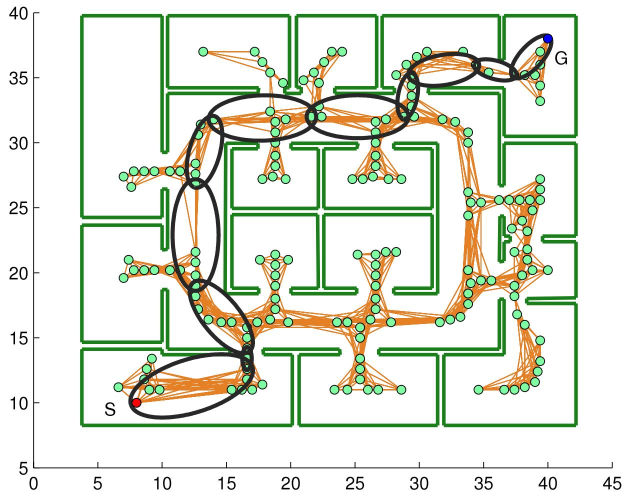

Fig. 6. Ellipses, $R = 5$ cells per metre, $P = 9$, solution with the least number of links.

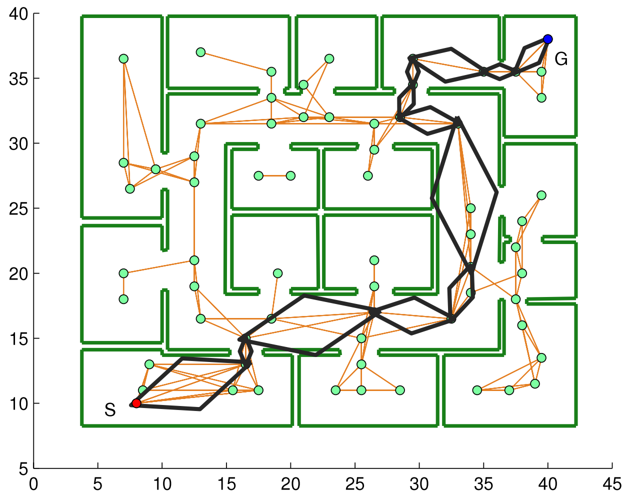

Fig. 7. Diamond shapes, $R = 2$ cells per metre, $P = 9$, solution with the least number of links.

decreased, which consequently decreases the maximum number M of *free space* cells, and (b) the maximum number of nodes N in

[3] The C++ implementation of Roaming Trails which has been used for experiments is freely available at the address www.robotics.laboratorium.dist.unige.it.

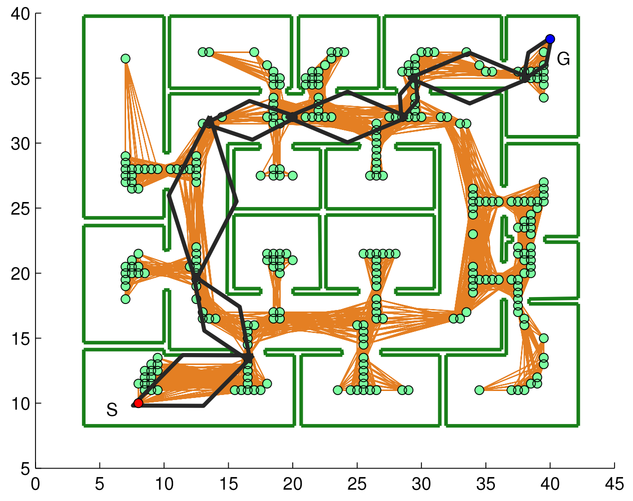

Fig. 8. Diamond shapes, $R = 2$ cells per metre, $P = 0$, solution with the least number of links.

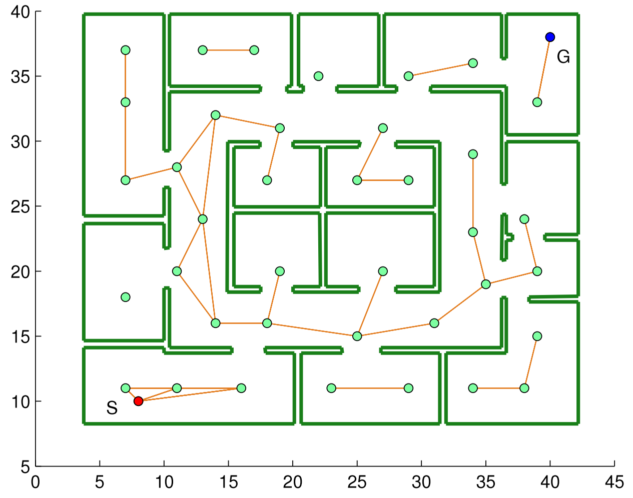

Fig. 9. Diamond shapes, $R = 1$ cells per metre, $P = 9$, no solution found.

the roadmap can be reduced by increasing P, which determines the minimum distance between roadmap nodes.

Fig. 7 shows the same problem as in Fig. 4, with a lower resolution $R = 2$, and $P = 9$: the times required are $t(\mathcal{G}) \approx 0.26$ s, $t(\mathcal{R}) \approx 0.18$ s, and $t(\mathcal{P}) \approx 0.006$ s. Fig. 8 shows the solution found by setting $R = 2$ and $P = 0$, i.e., without any constraints on the distance among nodes: the times required are $t(\mathcal{G}) \approx 0.26$ s, $t(\mathcal{R}) \approx 2.36$ s, and $t(\mathcal{P}) \approx 0.038$ s. Fig. 9 shows that, by setting $R = 1$ and $P = 9$, no solution can be found, since the resulting roadmap is made of disconnected components: the times required are $t(\mathcal{G}) \approx 0.068$, $t(\mathcal{R}) \approx 0.042$ s, and $t(\mathcal{P}) \approx 0.004$ s.

In order to quantitatively evaluate how the number of nodes and the computational times vary depending on R and P, a set of experiments have been performed in a very large environment with 10×10 rooms: each room is approximately 6 m $\times$ 6 m, and it is connected to neighbouring rooms. Table 1 summarizes results corresponding to the problem in Fig. 10 (roadmap nodes and links are not shown): rows are ordered according to the increasing number of nodes N, and report both the solution $\mathcal{P}_1$ with the shortest path $l(\mathcal{P}_1)$, and the solution $\mathcal{P}_2$ with the least number of links $l(\mathcal{P}_2)$. It can be noticed that, in this very large environment, a big number of roadmap nodes are produced, and the computational times $t(\mathcal{G})$, $t(\mathcal{R})$, and $t(\mathcal{P})$ increase accordingly: specifically, the time $t(\mathcal{R})$ increases with the square of N, up to $t(\mathcal{R}) \approx 17$ h in the last row of the Table. However,

Table 1
Computational time in environments with 100 rooms.

R	P	N	$t(\mathcal{G})$ (s)	$t(\mathcal{R})$	$t(\mathcal{P}_1)$ (s)	$t(\mathcal{P}_2)$	$l(\mathcal{P}_1)$	$l(\mathcal{P}_2)$
1	11	129	2.0	4.8 s	0.15	0.15 s	n.a.	n.a
1	9	182	2.0	9.5 s	0.21	0.22 s	n.a.	n.a
1	7	265	2.0	20.6 s	0.31	0.31 s	n.a.	n.a
2	11	360	7.9	38.6 s	0.43	0.43 s	n.a.	n.a
2	9	475	7.9	66.2 s	0.57	0.59 s	107.9	26
1	5	496	2.0	71.9 s	0.59	1.34 s	n.a.	n.a
2	7	755	7.9	3 min	0.91	0.92 s	97.8	27
5	11	1 074	48.9	6 min	1.28	1.34 s	88.3	19
5	9	1 609	49.1	12 min	2.10	2.04 s	86.3	19
1	3	1 619	2.0	13 min	1.9	2.1 s	88.8	22
2	5	1 667	7.9	14 min	2.0	2.18 s	87.2	22
5	7	2 426	48.8	29 min	2.9	3.5 s	85.4	19
5	5	5 039	48.9	2 h	6.2	12.7 s	83.6	19
2	3	5 835	7.9	3 h	7.3	15.9 s	84.1	19
5	3	14 486	48.9	17 h	22.2	3 min	83.4	19

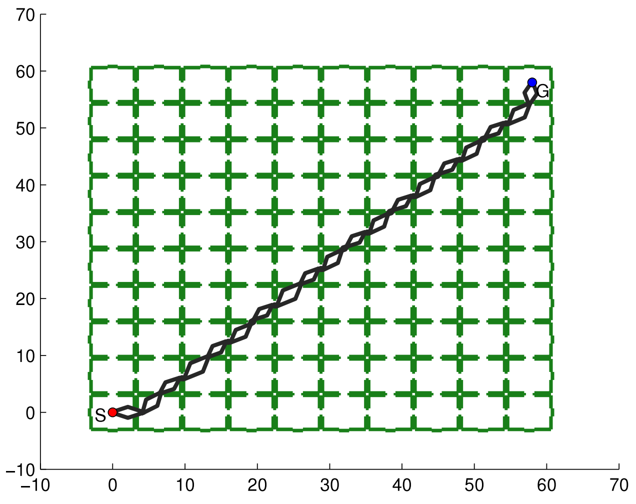

Fig. 10. Diamond shapes, $R = 5$ cells per metre, $P = 9$, least cost solution.

it is worth noticing that $t(\mathcal{R})$ correspond to steps 2, 3, 4 of Algorithm 1 which can be performed off-line, whereas the time $t(\mathcal{P})$ corresponding to on-line planning is much lower. Also, even if it can happen that a solution cannot be found when the number of nodes is too small (label *n.a.* in rows 1–4 and 6), increasing the number of nodes above a given threshold affects only marginally the length of the solution: this is particularly evident by inspecting the length $l(\mathcal{P}_2)$ of the solution with the least number of links. The solution shown in Fig. 10 corresponds to the 9th row of the table, with $t(\mathcal{G}) \approx 49.1$ s, $t(\mathcal{R}) \approx 12$ m, $t(\mathcal{P}_1) \approx 2.10$ s, and $t(\mathcal{P}_2) \approx 2.04$ s.

The approach has been extensively tested, for more than a decade, on all the robots of the Robotics Laboratory at DIST Università di Genova: primarily the autonomous robot Staffetta, a car-like platform with two rear motors and a front active steering wheel (a design choice which improves stability and reduces slippage) that we have designed for autonomous transportation within hospitals. Staffetta has a payload of about 120 kg and it is able to move at a maximum speed of about 1 m/s, it is equipped with proximity sensors for obstacle detection/avoidance and touch sensors for collision recovery, and it relies on a laser-based localization system to periodically correct its position in the environment. Fig. 11 shows still images from a video showing smooth obstacle avoidance thanks to the joint action of *Trajectory Generator*, *Local Map Builder*, *APF Builder* and *Safe Navigator*. When the front steering wheel is actively controlled, smooth motion capabilities help preserving kinematic coherency between the front and the rear wheels. Consider the following non-smooth motion behaviour, very frequent in reactive navigation

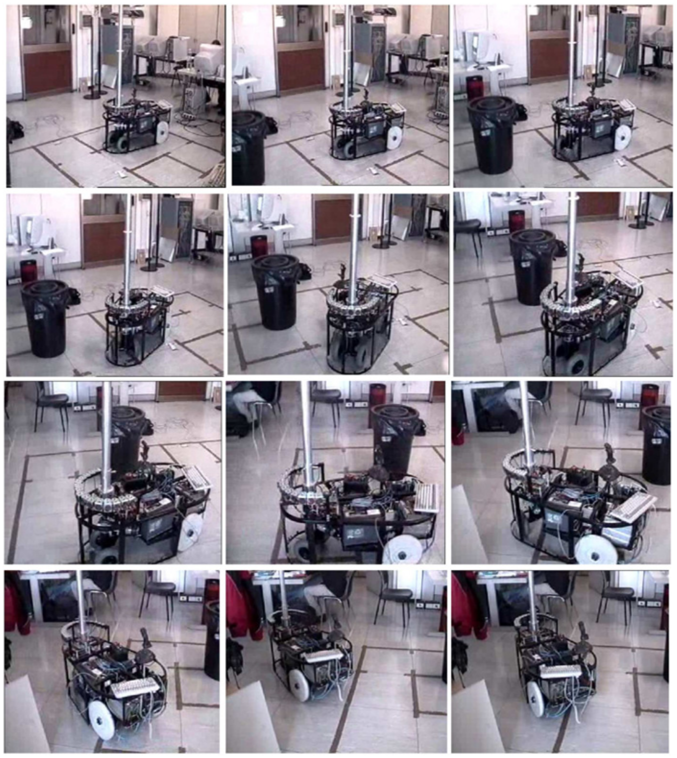

Fig. 11. Staffetta (chassis only) smoothly avoiding a static obstacle.

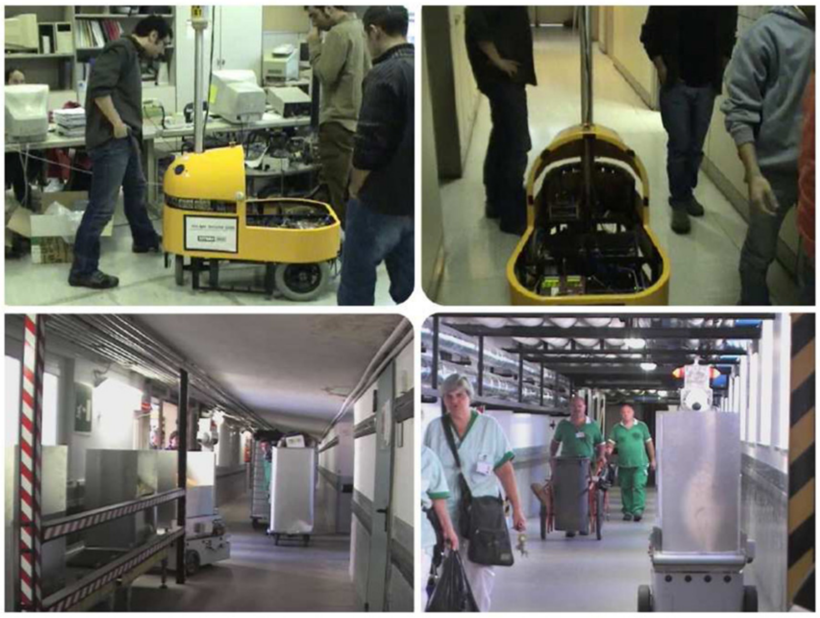

Fig. 12. Top: Staffetta at the 2nd floor of DIST. Bottom: Merry Porter^{TM} at Polyclinic of Modena, loading waste (left) and moving along a crowded corridor (right).

approaches: the robot is moving on a straight line, slow down in front of an obstacle, and starts searching for a safe direction of motion. In order to preserve kinematic coherency, every time the signs of the angular speed changes while the linear velocity has a small value above zero, an almost 180° rotation of the front steering wheel is required. Under these conditions, smooth motion is required to prevent the robot from spending all time adjusting the orientation of the front wheel.

Experiments have been performed at the 2nd floor of DIST Università di Genova (Fig. 12 on the top), at the Gaslini Hospital of Genoa and in public exhibitions, such as the Tmed 2001 exhibition (Magazzini Del Cotone, Porto Antico di Genova, October 2001). Moreover, thanks to the lesson learned with Staffetta, a second generation of robots have been developed (Merry Porter^{TM}, Fig. 12 on the bottom), and are currently being set-up within the Polyclinic of Modena for autonomous waste transportation.[4]

All the agents described in the paper are executed concurrently on board the robot, allowing to merge global planning with a

[4] Staffetta and the Merry Porter^{TM} are commercialized by the company Genova Robot srl, www.genovarobot.com.

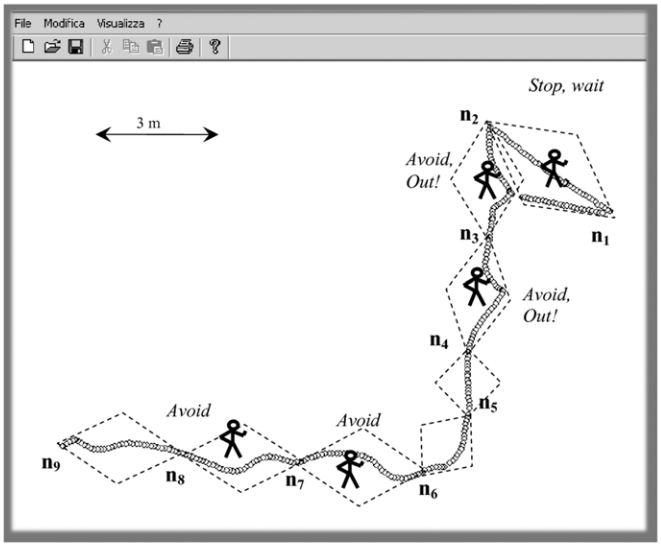

Fig. 13. Trajectory of the robot during one mission.

reactive, yet smooth and safe motion behaviour which takes into account the available a priori knowledge. Fig. 13 shows the trajectory followed by the robot while executing one mission: in some cases the robot has to stop and wait for persons to move on (labels "Stop" and "Wait"), whereas in some other cases it manages to avoid them (label "Avoid"). Finally, it can happen that the robot reaches the Roaming Trail boundaries while trying to avoid a person on its path (label "Out!"). There are no rough turns in the robot trajectory, except for the following cases: (a) a target node has been reached, (b) *Emergency Handler* intervenes either because the robot has reached the boundaries of the Roaming Trail, or an obstacle is too close to be smoothly avoided.

The experience gained in one decade of experiments with TRC Labmate, Staffetta and Merry Porter^{TM} allows us to draw the following qualitative conclusions concerning the usage of Roaming Trails in crowded indoor environments: robots are able to smoothly avoid approximately half of the obstacles; the remaining obstacles are dealt with by stopping, warning, and waiting for people to move on *or* to remove objects. Moreover, thanks to the presence of Roaming Trails – and given that an accurate localization system is available [34] – robots never reach deadlock situations from which it is hard to recover, yielding approximately a 100% success in navigation missions.

6. Conclusions

Experience shows that, when working within human environments, most existing systems are unable to work continuously with no failure or performance degradation. This is mainly due to the fact that human environments cannot be completely known a priory since they are highly dynamic, causing a dramatic effect both on self-localization and on navigation. It is not a case that the few examples of autonomous mobile robots which successfully hit the market have been cleaning robots and lawnmowers, i.e., robots in which superior navigation and self-localization capabilities are not required in the first place: since the task is simply to maximize the coverage of a given area in a statistical sense, a pre-planned path is not required at all, and navigation can be dealt with through a set of heuristic rules plus random noise, allowing the robot to change direction of motion when required. As an aside, notice also that the same approach would not work with other coverage tasks, such as painting or humanitarian de-mining: even if the task is similar, painting and humanitarian de-mining cannot be statistically faced: a 95% cleaned floor can be reasonably defined "a cleaned floor", whereas a 95% de-mined field or a 95% painted surface are definitely NOT a "de-mined field" or a "painted surface".

Then, in most robotics applications, superior navigation capabilities are required to plan and follow a path. The paper describes a two-level cognitive architecture which faces the problem by integrating, as usual, deliberative and reactive activities: the approach is original in that it introduces the concept of Roaming Trails. Instead of a path to the goal, the planner output is a Roaming Trail, i.e., a chain of diamond-shaped or elliptic areas which define the maximum allowed distance for the robot to deviate from its path to the goal. During mission execution, local navigation algorithms are allowed to reactively compute or simply update the robot trajectory, but always within the boundaries of the planned Roaming Trail. The approach has been tested for more than one decade on the robots TRC Labmate, Staffetta, and Merry Porter^{TM} in many different indoor environments, significantly improving the performance of the navigation system.

References

[1] F. Capezio, F. Mastrogiovanni, A. Sgorbissa, R. Zaccaria, The ANSER project: Airport nonstop surveillance expert robot, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007. IROS 2007, 2007, pp. 991–996.
[2] R. Vidal, O. Shakernia, H. Kim, D. Shim, S. Sastry, Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation, IEEE Transactions on Robotics and Automation 18 (5) (2002) 662–669.
[3] A. Sgorbissa, R. Zaccaria, The artificial ecosystem: a distributed approach to service robotics, in: 2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04, vol. 4, 2004, pp. 3531–3536.
[4] S. Berman, Y. Edan, M. Jamshidi, Navigation of decentralized autonomous automatic guided vehicles in material handling, IEEE Transactions on Robotics and Automation 19 (4) (2003) 743–749.
[5] A. Yamashita, T. Arai, J. Ota, H. Asama, Motion planning of multiple mobile robots for cooperative manipulation and transportation, IEEE Transactions on Robotics and Automation 19 (2) (2003) 223–237.
[6] R. Siegwart, K.O. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, R. Philippsen, R. Piguet, G. Ramel, G. Terrien, N. Tomatis, Robox at expo.02: a large-scale installation of personal robots, Robotics and Autonomous Systems 42 (3–4) (2003) 203–222.
[7] H. Moravec, Rise of the robots, Scientific American (1999) 124–135.
[8] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, The International Journal of Robotics Research 5 (1) (1986) 90–98.
[9] J. Borenstein, Y. Koren, The vector field histogram-fast obstacle avoidance for mobile robots, IEEE Transactions on Robotics and Automation 7 (3) (1991) 278–288.
[10] M. Piaggio, A. Sgorbissa, AI-CART: an algorithm to incrementally calculate artificial potential fields in real-time, in: 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation, 1999. CIRA '99. Proceedings, 1999, pp. 238–243.
[11] Y. Koren, J. Borenstein, Potential field methods and their inherent limitations for mobile robot navigation, in: 1991 IEEE International Conference on Robotics and Automation, 1991. Proceedings, 1991, vol. 2, pp. 1398–1404.
[12] R.C. Arkin, Motor schema based mobile robot navigation, The International Journal of Robotics Research 8 (4) (1989) 92–112.
[13] M. Slack, Navigation templates: mediating qualitative guidance and quantitative control in mobile robots, IEEE Transactions on Systems, Man and Cybernetics 23 (2) (1993) 452–466.
[14] S. Quinlan, O. Khatib, Elastic bands: connecting path planning and control, in: 1993 IEEE International Conference on Robotics and Automation, 1993. Proceedings, 1993, vol. 2, pp. 802–807.
[15] R.P. Bonasso, D. Kortenkamp, D.P. Miller, M. Slack, Experiences with an architecture for intelligent, reactive agents, Journal of Experimental and Theoretical Artificial Intelligence 9 (1995) 237–256.
[16] R. Simmons, The curvature-velocity method for local obstacle avoidance, in: 1996 IEEE International Conference on Robotics and Automation, 1996. Proceedings, vol. 4, 1996, pp. 3375–3382.
[17] D. Fox, W. Burgard, S. Thrun, The dynamic window approach to collision avoidance, Robotics Automation Magazine, IEEE 4 (1) (1997) 23–33.
[18] I. Ulrich, J. Borenstein, VFH*: local obstacle avoidance with look-ahead verification, in: IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA '00, vol. 3, 2000, pp. 2505–2511.
[19] O. Brock, O. Khatib, Elastic strips: a framework for motion generation in human environments, The International Journal of Robotics Research 21 (12) (2002) 1031–1052.
[20] L. Lapierre, R. Zapata, P. Lepinay, Simultaneous path following and obstacle avoidance control of a unicycle-type robot, in: 2007 IEEE International Conference on Robotics and Automation, 2007, pp. 2617–2622.
[21] A. Elfes, Using occupancy grids for mobile robot perception and navigation, Computer 22 (6) (1989) 46–57.
[22] I. Ulrich, J. Borenstein, VFH+: reliable obstacle avoidance for fast mobile robots, in: 1998 IEEE International Conference on Robotics and Automation, 1998. Proceedings, vol. 2, 1998, pp. 1572–1577.

[23] R.C. Arkin, T. Balch, Aura: principles and practice in review, Journal of Experimental and Theoretical Artificial Intelligence 9 (1997) 175–189.
[24] R. Kurozumi, T. Yamamoto, Implementation of an obstacle avoidance support system using adaptive and learning schemes on electric wheelchairs, in: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005), 2005, pp. 1108–1113.
[25] J. Minguez, The obstacle-restriction method for robot obstacle avoidance in difficult environments, in: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005), 2005, pp. 2284–2290.
[26] J. Minguez, Integration of planning and reactive obstacle avoidance in autonomous sensor-based navigation, in: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005), 2005, pp. 2486–2492.
[27] F. Zhang, A. O'Connor, D. Luebke, P. Krishnaprasad, Experimental study of curvature-based control laws for obstacle avoidance, in: 2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04, vol. 4, 2004, pp. 3849–3854.
[28] P. Ogren, N. Leonard, A convergent dynamic window approach to obstacle avoidance, IEEE Transactions on Robotics 21 (2) (2005) 188–195.
[29] B. Damas, J. Santos-Victor, Avoiding moving obstacles: the forbidden velocity map, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009, 2009, pp. 4393–4398.
[30] K. Arras, J. Persson, N. Tomatis, R. Siegwart, Real-time obstacle avoidance for polygonal robots with a reduced dynamic window, in: IEEE International Conference on Robotics and Automation, 2002. Proceedings. ICRA '02, vol. 3, 2002, pp. 3050–3055.
[31] M. Aicardi, G. Casalino, A. Bicchi, A. Balestrino, Closed loop steering of unicycle like vehicles via lyapunov techniques, Robotics Automation Magazine, IEEE 2 (1) (1995) 27–35.
[32] D. Soetanto, L. Lapierre, A. Pascoal, Adaptive, non-singular path-following control of dynamic wheeled robots, in: 42nd IEEE Conference on Decision and Control, 2003. Proceedings, vol. 2, 2003, pp. 1765– 1770.
[33] F. Lamiraux, D. Bonnafous, O. Lefebvre, Reactive path deformation for nonholonomic mobile robots, IEEE Transactions on Robotics 20 (6) (2004) 967–977.
[34] F. Mastrogiovanni, A. Sgorbissa, R. Zaccaria, Designing a system for map-based localization in dynamic environments, in: T. Arai, R. Pfeifer, T.R. Balch, H. Yokoi (Eds.), IAS, IOS Press, 2006, pp. 173–180.
[35] M. Piaggio, A. Sgorbissa, R. Zaccaria, Pre-emptive versus non-pre-emptive real time scheduling in intelligent mobile robotics, Journal of Experimental and Theoretical Artificial Intelligence 12 (2) (2000) 235–245.
[36] J.-C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, Norwell, MA, USA, 1991.
[37] K. Sugihara, Approximation of generalized voronoi diagrams by ordinary voronoi diagrams, CVGIP: Graphical Models and Image Processing 55 (1993) 522–531.
[38] P. Morasso, V. Sanguineti, Computational maps and target fields for reaching movements, in: Self-organization, Computational Maps, and Motor Control, in: Advances in Psychology, vol. 119, North-Holland, 1997, pp. 507–546.
[39] M. Piaggio, A. Sgorbissa, G. Vercelli, R. Zaccaria, Fusion of sensor data in a dynamic representation, in: Proceedings of the First Euromicro Workshop on Advanced Mobile Robot, 1996, 1996, pp. 10–16.

Dr. Antonio Sgorbissa received the Laurea degree in Electronic Engineering from the University of Genova, Italy, in 1996. In 2000 he received his Ph.D. in Robotics from the University of Genova, Italy, and from 2001 to 2004 he was a Post Doc at Georgia Tech, University of Parma and later at University of Genova. Since 2005 he has been Assistant Professor at the Department of Communication, Computer and System Sciences (DIST) of the University of Genova. He currently teaches Ambient Intelligence and Real-Time Operating Systems at the Faculty of Engineering, and Geographic Information Systems and Cognitive Robotics at the Faculty of Humanities. His main research interests are in the areas of mobile robotics, multi-robot systems, and ambient intelligence. His research interests also include planning, knowledge representation, and machine consciousness. He is the author/coauthor of more than 100 international scientific papers, and of two international patents in Robotics.

Renato Zaccaria is Full Professor in Computer Science at the Department of Communication, Computer and System Sciences (DIST), University of Genova, Italy. He leads the Mobile Robotics & Ambient Intelligence Research Group in the lab called Laboratorium. He has been a researcher and project leader in many national and European projects in Autonomous Robotics, and has been responsible of many industrial contracts as well. His present activity includes mainly Service Robotics and Ambient Intelligence. In order to purposively carry out technology transfer, in 2000 he founded one of the Department's spin-off companies, Genova Robot, whose present R&D activity is strongly linked to academic research. He is the author/coauthor of more than 140 international scientific papers, of two textbooks, and of two international patents in Robotics (guidance of autonomous mobile robots). Presently, he is responsible of the European Master on Advanced Robotics EMARO, the coordinator of the Robotics Engineering Master Track, and the head of the Commission for the Study Programmes at the Faculty of Engineering. He has been awarded the title Commendatore della Repubblica because of his activity in technology transfer in Robotics.