Journal of Ambient Intelligence and Smart Environments 0 (0) 1
IOS Press

Efficient customisable dynamic motion planning for assistive robots in complex human environments [1]

Alessio Colombo [a,*], Daniele Fontanelli [b], Axel Legay [c], Luigi Palopoli [a] and Sean Sedwards [c]
[a] *Department of Engineering and Computer Science, University of Trento, Italy*
E-mail: {colombo,palopoli}@disi.unitn.it
[b] *Department of Industrial Engineering, University of Trento, Italy*
E-mail: {daniele.fontanelli}@unitn.it
[c] *INRIA Rennes – Bretagne Atlantique, France*
E-mail: {axel.legay,sean.sedwards}@inria.fr

Abstract. People with impaired physical and mental ability often find it challenging to negotiate crowded or unfamiliar environments, leading to a vicious cycle of deteriorating mobility and sociability. To address this issue we present a novel motion planning algorithm that is able to intelligently deal with crowded areas, permanent or temporary anomalies in the environment (e.g., road blocks, wet floors) as well as hard and soft constraints (e.g., "keep a toilet within reach of 10 meters during the journey", "always avoid stairs"). Constraints can be assigned a priority tailored on the user's needs. The planner has been validated by means of simulations and experiments with elderly people within the context of the DALi European project.

Keywords: Motion planning, Assistive robotics, Mapping

1. Introduction

With unimpaired ability, pedestrians are able to find their way across complex and crowded areas with few problems. With reduced abilities this apparently simple task easily becomes a challenging one. A person with reduced mobility needs to minimise the travelled distance. A person with cognitive problems should avoid situations that challenge her sense of direction and confuse her perception of the environment. The difficulty in identifying the best path and in making proper reactions to unexpected contingencies gradually reduce the confidence of the impaired person in using public spaces. The afflicted are most often older adults and the problem worsens quickly if no adequate countermeasure is taken [3,37]. They are deprived of essential social relations with a negative impact on their physical condition (reduced exercise), on their psychological wellbeing (reduced social contact) and even on the quality of their nutrition when they reduce the frequency of their visits to supermarkets [35,1]. The application of assistive robotic technologies can be of significant help to offset this trend.

In this work we therefore consider a dynamic motion planning problem in a complex but known environment containing other moving agents (i.e., pedestrians). The planner's motion is constrained by the user's preferences (preferred speed, preferred proximity to others and preferred or disliked areas in the environment) and must accommodate any recent modifica-

[1] The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° ICT-2011-288917 "DALi - Devices for Assisted Living", and from the European Union's Horizon 2020 Research and Innovation Programme - Societal Challenge 1 (DG CONNECT/H) under grant agreement n° 643644 "ACANTO - A CyberphysicAl social NeTwOrk using robot friends".
[*] Corresponding author. E-mail: colombo@disi.unitn.it.

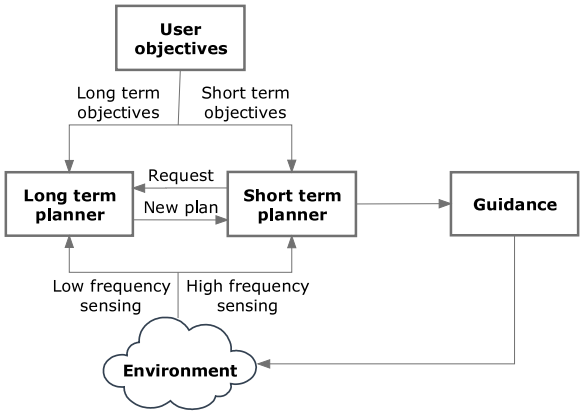


Fig. 1. Diagrammatic overview of the motion planning framework. The whole process can be divided into three main elements: the *long term planner* that considers the long term objectives, the *short term planner* that optimises the long term plan taking into account the short term objectives and constraints, and the guidance that drives the robot towards the goal.

tions to the environment that are not known a priori (e.g., due to maintenance work). The goal of the planner is to help the user visit a number of points of interest in the most stress-free and efficient manner. Our basic approach is to construct a graph data structure of sufficient granularity to represent efficient direct paths between points of interest. The planner attempts to follow paths defined on the graph, but may deviate to avoid other pedestrians or at the specific request of the user. The complexity of the problem is essentially that of finding the shortest path on a graph.

Figure 1 illustrates the different types of support that a robotic system with cognitive abilities can offer in the navigation of a complex environment.

From a top-down perspective, the first type of assistance is offered before taking on the navigation activity and is the production of a plan that takes into account long term objectives. This is accomplished by the *long term planner*, which accounts for the topology of the space, the user's preferences and the possible presence of obstacles or problems along the way, as foreseen by querying environmental sensors. While the user is moving, she could encounter contingent problems that cannot be anticipated (e.g., a small group of people obstructing the path). In this case her robot assistant could react by planning a minimal deviation from the path that preserves her safety and wellbeing. In our terminology, this component is called the *short term planner* (see Figure 1). Finally, a robot assistant can guide the user along the planned path. This can be done in different ways depending on the type of robot assistant. If the robot is simply a guiding vehicle, like a tour-guiding vehicle [33], guidance amounts to following the path and to ensuring that the user trails behind. If the robot is a robotic walker, it can guide the user by mechanically turning its wheels and acting on the wheel brakes [15] or by administering visual, audio or haptic signals [31,32]. If the robot is a robotised wheelchair it can be assimilated to a robotic vehicle driving in crowded spaces [4].

This paper proposes a novel *long term planner* that targets the first task described above. Consider a person willing to execute a set of activities in a public space. Henceforth, we will use the term assisted person (AP) to define the user. The problem the AP faces is to identify the best way to reach the chosen point of interest. This decision could potentially be taken using any state-of-the-art algorithms for motion planning, able to identify the path with minimum length (or requiring minimum time) given the a priori knowledge of the map. A first problem is that while the position of most fixed objects (e.g., buildings, rooms, and points of interest) is known a priori, the algorithm must account for the possibility of incidental changes, such as temporary obstructions. Standard motion planning algorithms can easily be adapted to consider an up-to-date picture of the state of the environment (e.g., presence of obstructions or over-crowded spaces) as it arrives from environmental sensors. However, a simple modification to a standard planner could be insufficient. First, the detected anomaly could be a temporary one. So, the likelihood of having to deal with the problem during the navigation depends on the time needed to reach the place where the anomaly is located, which in turn depends on the chosen path. What is more, the AP (who is typically an older adult) will likely have specific additional requirements. For instance, the AP could need a frequent access to the toilet, and if the optimum path offers no easy access to the toilet on the way, it could easily generate discomfort. The AP could be hyper-vigilant and overly concerned with her personal security. In this case, she might appreciate always being within reach of a policeman or of other staff members that she perceives as a reassuring presence.

Simply put, what we need is an algorithm for motion planning in public spaces that accounts for 1. the topological and metric information about the space, 2. time-varying environmental information about the space, such as the availability of services (is the shop that the AP wants to visit actually open?), the presence of occlusions and overcrowded areas, etc., 3. prefer-

ences of the AP (e.g., the need to be in easy reach of assistance, toilets, etc.).

The presence of these specific requirements makes the planning algorithms offered by commonplace navigators (such as Google Maps) infeasible. What is needed is a different approach that carefully considers the strong psychological aspects involved in the selection of a route. In this paper we report on the solution that we have developed within the context of two research initiatives sponsored by European Union: Devices for Assisted Living (DALi) and A CyberphysicAl social NeTwOrk using robot friends (ACANTO). These projects are based on a substantial involvement with users, both for requirements collection and for the evaluation of the results. The algorithm proposed here distils this experience and translates it into a suitable formal model by using a modified Dijkstra's shortest path algorithm [11] where the underlying graph is constructed using quad tree decomposition of the free space [5].

The paper is organised as follows. A review of the current state of the art is presented in Section 2, while a complete description of the requirements of the planner can be found in Section 3. Section 4 goes into the details of the planning algorithm and Section 5 illustrates the functionalities of the software architecture. Sections 6 and 7 report the results of qualitative and quantitative simulations respectively, and Section 8 describes our case study and the related experiments. Finally, in Section 9 we offer our conclusions.

2. Related work

Motion planning in crowded environments is a relevant research problem in robotics which has received constant attention throughout the past two decades [25, 27,24]. The approach that we advocate is based on a hierarchical decomposition of the problem between short term and long term planning. Different authors in the literature propose a strategy of this kind [28,18], but the solutions at each of the two levels of the hierarchy differ significantly, depending on the requirements that each author considers. The goal of a *long term planner* is to find an efficient path in free space from a starting point to some desired destinations, given the topological and metric constraints derived from the map. In the following sections we will compare the different components that compose the *long term planner* with respect to the state of art.

2.1. Shortest-path planning

2.1.1. Sampling methods

When the map is not entirely known in advance (e.g., due to uncontrollable changes in the environment), a convenient choice can be the adoption of sampling-based algorithms. In this class the Probabilistic RoadMap (PRM) algorithm by Kavraki et al. [21] and the Rapidly Exploring Random Trees (RRT) [26] have gained an undisputed reputation and visibility in the past few years. The idea of this class of algorithms is to generate feasible points by sampling randomly the neighbourhood of known points and connecting them into a data structure (e.g., a tree for RRT or a graph for PRM). When the destination is finally reached an optimal path can be found by exploring the data structure. The more time that is given to the computation, the more points that can be added and the higher the probability becomes of finding an optimal solution. Such algorithms have recently been revisited by Karaman and Frazzoli [20]. The revised versions, PRM* and RRT*, are probabilistically complete, meaning that if the algorithm is given enough time to explore the space, it eventually identifies the optimal solution with probability 1. An important point of these algorithms is that while the data structure is being created it is possible to enforce a hierarchy of hard and soft constraints penalising (or ruling out) points that would violate them. This is an appealing feature for us because our problem is characterised by a set of constraints. However, the construction of the map on the fly is an unnecessary computational burden in our case. Our intended operational scenario is a public space (e.g., a mall or a museum) for which a large amount of a priori information is usually available.

2.1.2. Potential fields methods

Another family of algorithms is based on the definition of potential fields [36,8] around obstacles and points of interest that can attract or repel the robot. Such approaches are known to be effective for obstacle avoidance, but they are often plagued by local minima (which sometimes delay or deadlock progress). While encoding all the user's planning requirements, constraints and preferences with a potential function is generally a difficult problem, our approach makes use of the notion of gradients to encode user-defined desirable and undesirable zones. The full details are given in Section 4.3

2.1.3. Graph based methods

The *long term planner* proposed in this work falls in the class of graph based techniques. In essence, the idea is to decompose the environment into a grid and then generate a graph by associating nodes to elements of the grid and by then connecting with edges the nodes associated to adjacent cells. Minimum time paths on the graph can be found using the well-known Dijkstra algorithm [11] or its extension A* [19]. The use of a constant size grid is generally discouraged due to the explosion of the configuration space size, hence several more efficient ways to construct the graph have been proposed. Possible approaches include Voronoi diagrams [2] and PRM [21]. We follow Chen et al. [5] and construct a graph using quad tree decomposition of the space, exploring it with an extended version of the Dijkstra algorithm. The generation of our quad tree is specific for its application to structured indoor spaces, with large rooms connected through corridors, doors and passageways and where each room may contain such things as counters, shelves and exhibition paraphernalia that compromise its regularity.

2.2. Time-dependent paths

The most important feature of our algorithm is its ability to deal with temporary anomalies (e.g., obstructions or large groups of people hindering the AP's motion across some of the areas). In particular, anomalies (i.e., temporary graph obstructions) require the generation of time-dependent paths. The underlying graph exploits a dynamic and time dependent cost function, thus the shortest paths between two edges can vary over time. This problem is known to be challenging and is the focus of independent research [12,9,10,16]. An interesting analysis on the complexity of this problem has been carried out by Foschini et al. [16]. They upper bounded the cost of traversing a graph with polynomial-size piecewise linear cost functions and with other particular classes of linear functions. However, the cost remains high and prohibitive even with small to medium sized graphs. In [9] the authors present an overview of existing techniques together with three efficient speed-up methods. The experiments, nevertheless, show that finding the solutions requires several minutes (sometimes hours) using server-class hardware.

Given these performance limitations, in our particular case we adopt a conservative assumption, described in Section 4.6, that allows us to solve a simplified problem very efficiently. Our requirement analysis reveals that senior users of a navigation tool become very annoyed by a long wait in front of a screen. Therefore, efficiency and quick deliveries of decisions are more important than producing "optimal" decisions (as long as the decisions do not violate any hard constraints and they respect soft constraints to a reasonable extent).

2.3. Global constraints and preferences

The global constraints (not to be confused with kinodynamic constraints, not considered here) are used for customising the behaviour of the planner and for introducing the notion of "comfort" for the AP. Constraints are prioritised and some of them can be violated if their compliance prevents the system from finding any path. They embed priority and the possibility for one or more constraints to be ignored if a path cannot be found otherwise (namely, conflicting constraints). This is called "planning with partial satisfaction", and is studied in the literature under the notion of *preference-based planning*. In [6] the authors focus on computation of relaxed plan-based heuristics that guide the planner towards good solutions satisfying the given preferences expressed in Planning Domain Definition Language (PDDL). PDDL is one of the languages aimed at standardising Artificial Intelligence planning and is used in many international competitions. However, its complexity and completeness are an overkill with respect to the goals of this work.

A growing set of frameworks in the literature [34, 23] proposes to express temporal properties with partial satisfaction using linear temporal logic (LTL). In [23] the authors introduce a method for quantifying the satisfaction of LTL formulae, and propose a planning framework that synthesises robot trajectories with the optimal satisfaction value. However, they do not consider constraints where the cost or priority changes over time. Tumova et al. [34] present an automatic generator for control strategies for a robotic vehicle where constraints are expressed with LTL formulae. The novelty is the possibility of violating a constraint, according to its priority, in order to complete the task (e.g., a road lane should not be crossed, but this is allowed during car parking).

The concept of "comfort" has already appeared in the literature with different meanings: 1) comfort of the AP when navigating using a robotic platform [17, 29] and 2) comfort of the humans in the area surrounding an autonomous robot [22]. Our notion of comfort belongs to the first class and it is deeply rooted in the requirement analysis and in the validation activities

with senior users that we have been conducting in the context of the DALi and of the ACANTO projects. Our findings are that the AP needs to specify zones that she likes or dislikes. As an example, more often than not she would prefer to bypass crowded areas or to always have a toilet or a resting place within easy reach, even if this entails choosing a slightly longer path.

3. Requirements, preliminaries and overview

The proposed *long term planner* has been developed bearing in mind a number of requirements. The key point is letting the AP personalise her journey while keeping the planner reactive to changes in the environment. For this reason we have implemented three main features.

The first feature gives the AP the possibility of adding hard (non-violable) and soft (violable) constraints, according to some customisable priority. It is possible to encode rules like "never get closer than 5 meters to any stair" or "try to keep within 10 meters of a toilet", or "always be within sight of a clerk or of a policeman". Should a soft constraint be in conflict with another one, the issue is resolved by violating the one with lower priority. A hard constraint, instead, cannot be violated. If the *long term planner* encounters an inconsistent state (e.g., not all hard constraints can be satisfied) then the AP is notified and is asked to review the set of constraints.

The second feature reacts to anomalies detected in the environment by the sensing subsystem. An anomaly is a bounded zone in the environment that becomes inaccessible for a limited period of time (e.g., a wet floor or blocked passage). After this period expires, the anomaly is cleared and the zone is accessible again.

The final feature takes into account the crowded spots in the environment. They are represented as heat maps (an example is shown in Figure 7) where the apparent "heat" represents the level of crowdedness. The planner interprets this level as a penalising factor that slows down the AP. Some users could also have specific constraints related to avoiding crowded areas.

The work flow of the algorithm begins with the AP specifying a list of target locations she wants to visit in the environment. The *long term planner* constructs a plausible path (a long term plan) according to the constraints in her profile and to the current conditions in the environment (known anomalies and current crowding represented by heat maps). This data is sampled periodically from remote sensors (e.g., surveillance cameras). If other robots are deployed in the environment, they can use their local sensing system to detect anomalies and share this information through a cloud infrastructure. For instance, if a walker detects a wet floor sign, this information is propagated to the other robots and accounted for in the generation of long term plans. Once the AP accepts the plan and starts moving, the control subsystem takes over, allowing the short term planner to make limited adjustments depending on the contingencies encountered on the ground. In the event that the AP is unable to follow the plan with only such limited modifications (e.g., because of encountering an unforeseen obstacle), the control subsystem has the capability to report the event and can request the construction of a new long term plan.

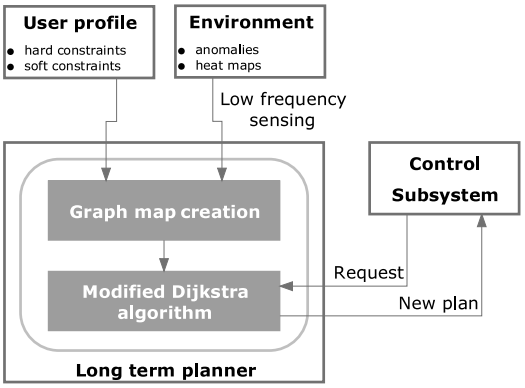


Fig. 2. Diagrammatic overview of the *long term planner*. Informed by the heat maps and anomaly detectors, the *long term planner* constructs a long term plan according to the AP's constraints. The plan is then transferred to the control subsystem.

The *long term planner* produces the optimal path according to the diagram depicted in Figure 2 and described as follows:

1. the plan of an environment is broken down into a grid of rectangular cells containing free space
2. a graph is derived from the grid, such that each node is on the border between two cells and each edge defines a path in free space
3. nodes corresponding to points of interest that are not already present are added manually
4. the graph is augmented with relevant semantic information (e.g., associating the names of points of interest to nodes)
5. each edge is associated with a cost that accounts for the distance to travel and for the occupancy of the area (the more people, the longer the time to travel)

6. parts of the graph are removed or the weights of edges increased to reflect the AP's preferences
7. the optimal path is found using a modified Dijkstra algorithm.

In the next sections the algorithm is presented in its full details.

3.1. Preliminaries

To describe our long term planner, we first define some notation and operations on graphs.

A directed graph $G = (N, E)$ is a set of nodes $n \in N$ linked by a set of edges $e \in E$. An edge $e = (n, n') \in E$ is defined by its two adjacent nodes $n, n' \in N$.

Given graphs $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$, $G_1 \subseteq G_2 \implies N_1 \subseteq N_2 \wedge E_1 \subseteq E_2$ means that G_1 is a subgraph of G_2.

Given $G_1 \subseteq G_2$, $G_2 \backslash G_1 = (N_2 \backslash N_1, E_2 \backslash \{e \in E_1 \mid e = (n, n') \wedge (n \in N_1 \vee n' \in N_1)\}$ is the graph that remains after removing G_1 from G_2. We do not consider $G_2 \backslash G_1$ if $G_1 \not\subseteq G_2$

Pairwise graph union is defined by $G_1 \cup G_2 = (N_1 \cup N_2, E_1 \cup E_2)$. The union of a set of graphs $\mathcal{G} = \{G_1, G_2, G_3, G_4, \ldots, G_m\}$ is denoted $\bigcup \mathcal{G}$ and performed pairwise, such that $\bigcup \mathcal{G} = ((\cdots(((G_1 \cup G_2) \cup G_3) \cup G_4) \cup \cdots) \cup G_m)$.

4. Long term planner

The *long term planner* proposes feasible paths that efficiently visit the AP's specified points of interest, while respecting her preferences and accommodating the prevailing conditions in the environment. To achieve this, the *long term planner* abstracts a complex environment, such as a shopping mall, airport, museum, etc., as a weighted directed graph, comprising a set of *nodes* linked by *edges*. The nodes represent places in the environment, while the edges represent direct paths between the places and are weighted by their *effective length*. The a priori length of an edge is the Euclidean distance between its adjacent nodes. The effective length of an edge is generally longer, modelling its undesirability with respect to crowding and the AP's preferences. The edges are directed so that the effective length of a path leading to an undesirable area can be greater than the same path traversed in the opposite direction.

Nodes are labelled with their physical location (coordinates on the plan of the environment) and their corresponding semantic position (*supermarket*, *toilet*, *post office*, *café*, *bar*, *bakery*, etc.). Each edge in the graph is labelled (weighted) with the effective distance between its adjacent nodes. Then, using an efficient graph traversal algorithm, i.e., the Dijkstra algorithm [11], we find the shortest paths that link the AP's points of interest. Moreover, anomalies and crowding are included in the same framework by simply modifying the graph prior to finding the shortest path. In particular, anomalies cause parts of the graph to be (temporarily) removed, while crowding increases the weights of edges in crowded areas (their *effective* length is increased because crowding slows the AP's progress). Certain user preferences, such as always being near a toilet, may also be encoded as graph transformations.

4.1. Creating graphs from floor plans

To construct a graph that efficiently maps the free space in the environment, we first decompose its floor plan into a 'quad tree' [14], comprising quadrants containing free space (free quadrants) and quadrants occupied by fixed objects (occupied quadrants). A graph is constructed by embedding nodes in only the free quadrants and linking them with appropriate edges. The quad trees typically have substantially fewer cells than a uniform grid with the same level of minimum granularity, with the density of cells generally following the density of features [13]. An example is shown in Figure 5. Note that the side length ratio is common to all quadrants and is inherited from the dimensions of the environment. It is possible to add space to the environment to force quadrants to be square or have any other desired ratio. Doing so may improve efficiency or be advantageous with respect to the placement of nodes.

Given a quad tree decomposition of the free space, the corresponding graph is constructed as follows. For all pairs of adjacent free quadrants, a node is embedded at the mid point of the border of the smaller of the quadrants. By definition, a free quadrant is a convex shape containing only free space. Hence, any node on the border of a free quadrant has a "line of sight" to all other nodes on the borders of the same quadrant. We therefore join such nodes with a complete graph. Since nodes are shared between adjacent quadrants, this is sufficient to link all the free space in the environment.

To guarantee that the robotic platform may occupy any point in free space represented by a node, or travel the line represented by any edge, prior to building the quad tree the fixed objects are enlarged in all directions by a distance greater than the radius of the robotic plat-

form. In this way no point in the *effective* free space is ever too close to a fixed object and all paths in the graph correspond to plausible paths in the environment.

4.2. Creating a long term plan

To represent the a priori knowledge about the environment we define a "graphmap" data structure $\mathcal{M} = (G, W, C, L)$. $G = (N, E)$ is a graph of the environment derived from a quad tree, as described in Section 4.1. Function $W : E \rightarrow (0, +\infty]$ assigns a length (the Euclidean distance between the points denoted by adjacent nodes) to all the edges of the graph. Function $C : N \rightarrow (\mathbb{Q}, \mathbb{Q})$ labels each node with its spatial coordinates in the environment. Function $L : N \rightarrow P \cup \{uninteresting\}$ labels each node with its semantic location, where $P = \{$*supermarket*, *bakery*, *café*, etc.$\}$ is a set of points of interest.

To generate a long term plan we also define a "working copy" of the graph map (the working graphmap), modified according to the AP's constraints, the current crowding and the known anomalies. We denote the working graphmap $\mathcal{M}' = (G' = (N', E') \subseteq G, W', C, L)$. In general, the graph G' excludes any inaccessible subgraphs arising from anomalies or the AP's constraints. The weighting function W' assigns an *effective* length to all edges, which includes the effects of crowding and the AP's constraints. The construction of G' and W' are described in Sections 4.3, 4.4 and 4.5.

Given a working graphmap $\mathcal{M}'$ and a (possibly ordered) set of user-specified points of interest, the *long term planner* proposes a path that visits the points of interest while respecting the AP's global constraints. Formally, given a user-specified set of points of interest $\{p_j \in P\}_{j=1}^{m}$, the planner suggests a path $\{n_i \in N'\}_{i=1}^{k}$ s.t. $\forall p_j \in \{p_1, \ldots, p_m\}\ \exists n_i \in \{n_1, \ldots, n_k\} \wedge L(n_i) = p_j$. If the path must respect the order of the specified points of interest, then additionally $\forall p_s, p_t \in \{p_1, \ldots, p_m\}, \nexists n_i, n_j \in \{n_1, \ldots, n_k\}$ s.t. $s > t \wedge i < j \wedge L(n_i) = p_s \wedge L(n_j) = p_t$ holds true.

Finding the minimum length path that visits a set of unordered points of interest is an instance of the well known NP-hard 'travelling salesman problem' [30]. Moreover, given that the overall excursion (including stops at the points of interest) may take considerable time, an overall plan optimised for the current level of crowding may eventually be significantly suboptimal if the crowds dissipate. Our approach is therefore to optimise each leg of the journey separately, using the most up-to-date information about anomalies and crowding.

In simple terms, long term planning works in the following way. The planner first identifies the node $n_0 \in G'$ that is closest to the AP's current coordinates (x_0, y_0). This is given by $n_0 = \arg\min_{n \in G'} \parallel C(n) - (x_0, y_0) \parallel$. If the AP's points of interest have been specified in order, the planner uses Dijkstra's algorithm to find the shortest path between n_0 and the next unvisited point of interest specified by the AP. If the AP has not specified an order, the planner uses a modification of Dijkstra's algorithm to find the shortest path between n_0 and the closest unvisited point of interest. Given the trajectory and the AP's coordinates, n_0 may not be the optimum first node in the path (it may be effectively behind the AP on the path). The planner therefore sets the first node of the path to be the node by which the AP will leave the current quadrant.

The inclusion of time-dependent anomalies makes the actual long term planning algorithm slightly more complex. Handling such anomalies is described in Section 4.5.

4.3. Global constraints

The AP may specify constraints that affect the long term plan (e.g., always remain within 50 metres of a toilet). We call these "global" constraints to distinguish them from, for example, local constraints that might be implemented by the short term planner (e.g., don't get too close to other pedestrians). Global constraints may be hard or soft. Hard constraints exclude parts of the environment that the AP does not wish to visit under any circumstances. They are implemented by removing subgraphs from G. The set of hard constraints is denoted $x \in X$, where $x \subseteq G$ and two hard constraints $x, x' \in X$ are not necessarily disjoint. Hence $G' = G \backslash \bigcup X$. Removing parts of the graph may significantly lengthen the planned journey or make it impossible, hence the final plan (or lack of it) is presented to the AP for approval.

Soft constraints make parts of the environment desirable or undesirable to the *long term planner*, causing the planned path to deviate towards or away from them, respectively. They are implemented by defining a function $K : E \rightarrow [1, +\infty]$ that modifies the weights of edges to and from desirable and undesirable nodes. The function K is applied according to (1), introduced in Section 4.4. If no constraint applies to the nodes adjacent to edge e then $K(e) = 1$. In gen-

eral, given two nodes n and n' connected by edges $e = (n, n')$ and $e' = (n', n)$, for a single constraint $K(e) > K(e') \iff n$ is more desirable than n'. In the case of multiple constraints applying to the same edge e, the value of $K(e)$ is the maximum considering all constraints. We adopt this approach to avoid the situation that two constraints effectively cancel each other and also because our interpretation of desirability is that it is not simply additive.

In our implementation, soft constraints are specified using sets of triples (*location*, *radius*, *intensity*), which respectively define the semantic position, the radius of influence and the intensity of the constraint. In general, a constraint creates a gradient of weights that increase towards undesirable zones and vice versa for desirable zones.

A function $\tilde{K}_i : [0, radius] \rightarrow [1, intensity]$ is defined that maps distance from the border of location i to the weight of the gradient. This function should be monotonic non-increasing in case of undesired locations, and monotonic non-decreasing in case of desired locations. In both cases its integral should be finite (i.e., the *radius* of influence should be finite). Function $\tilde{K}_i$ is later used by $K(e)$ for associating the weight to each edge. It is worth noting that there is high flexibility in the choice of $\tilde{K}_i$, which improves the expressiveness of global constraints, allowing per-user customisations (e.g., the profile of attraction to toilets might be different across users) as well as location based personalisation (e.g., the profile of repulsion of an open window is different to that of a flight of stairs).

We assume the existence of a set of constraints $s \in S$. The *location* of each constraint defines corresponding sets of either desirable or undesirable nodes $N_s \subseteq N$ that are not necessarily disjoint. Let $d(i, j)$ denote the minimum Euclidean path distance from node i to node j, then for any edge $e = (n, n') \in E$, the value of $K(e)$ is given by

$$K(e) = \max_{\forall s \in S} \left(K^*_{location_s} \left[\min_{n'' \in N_s} \left(d(n'', n') \right) \right] \right)$$

where $K^*_{location_s}$ is defined as:

$$K^*_{location_s}(r) = \begin{cases} \tilde{K}_{location_s}(r) & \text{if } r \in [0, radius] \\ 1 & \text{otherwise} \end{cases}$$

4.4. Heat maps

Cameras in the environment monitor pedestrian traffic and construct "heat maps" that estimate average occupancy of the free space over useful time periods, such as the last five minutes, the last hour or a long-term average for a particular day and time. The goal is to use this information to predict the crowdedness that the AP will encounter and to plan accordingly. In this work we assume that the current prediction is valid over the time the AP takes to reach the next point of interest. If future experience in real environments suggests this assumption is unreasonable, we will treat crowdedness in the same way we treat anomalies, i.e., as time-dependent.

Each point in the free space is thus assigned a value in the interval $[0, 1]$, denoting its time-averaged occupancy density. A point with average density 1 is effectively impassable. In practice, not all areas are monitored and monitored areas will be divided into an array of square cells of uniform local density. Unmonitored areas are assumed to have zero density. A camera's view may also include areas occupied by fixed objects, but such areas are not accessible by any edge of the graph and their density is therefore not used.

An edge represents a straight line path between the points in free space represented by its adjacent nodes. The average occupancy in the area surrounding the line affects the time taken to travel from one end to the other. The free space that the *short term planner* will allow the AP to explore can be approximated by an ellipse whose vertices ("ends") coincide with the ends of the line. The area of the ellipse represents the capacity of the edge, while the heat within the ellipse represents the amount of capacity that is being used by others. To calculate the average occupancy of an edge, we integrate the occupancy density over its corresponding ellipse. The size and shape of the ellipse is a function of the edge. For simplicity we define an occupancy function $H : E \rightarrow [0, 1]$ that implicitly includes knowledge of the current heat map and performs this integration. The *effective* length of an edge is then given by the function $W' : E \rightarrow (0, +\infty]$, defined as

$$W'(e) = \frac{K(e)W(e)}{1 - H(e)} \qquad \forall e \in E. \tag{1}$$

The intuition behind (1) is that the effective length of an edge e is proportional to the desirability $K(e)$ of the destination node and inversely proportional to the occupancy $H(e)$. When there is zero occupancy,

$H(e) = 0$ and the effective length is only related to the desirability $K(e)$ and the Euclidean distance $W(e)$. With full occupancy, $H(e) = 1$, the effective length is infinite and (1) correctly models the fact that the edge is impassable.

4.5. Anomalies

During the course of a journey the AP may encounter *anomalies* (semi-permanent obstructions, such as a wet floor, locked exit, dense crowd, etc.) that prevent the *short term planner* from making progress along the long term plan. An anomaly is represented by a data structure $(g \subset G, t \in (0, +\infty])$, where $g \subset G$ represents the inaccessible region of the environment and t is the estimated remaining time that the anomaly will last. The set of active anomalies (those with remaining time > 0) is denoted $a \in A$. Anomalies are removed from A when their remaining time reaches 0.

Anomalies exclude parts of the environment, but their effect is not permanent and is dependent on the chosen path. When a new anomaly (g, t) is detected by the short term planner, it is added to the set of active anomalies and its subgraph is immediately removed from the working graphmap. Symbolically, $A \leftarrow A \cup (g, t)$ and $G' \leftarrow G' \backslash g$. The shortest path to the next point of interest is calculated according to the procedure described in Section 4.2. The approximate time of reaching every node in the proposed path is calculated according to the average speed of the AP.

The new trajectory definitely excludes the recently detected anomaly, but may include one or more anomalies in A. Hence, the proposed plan is compared to the subgraphs in the set of active anomalies, to find if there is any intersection. If there is no intersection the proposed plan is valid. If the proposed trajectory intersects the subgraph of an anomaly, the time of reaching the anomaly is compared to its remaining time. If the anomaly will not exist by the time the AP reaches it, it is ignored. If no anomalies exist by the time the AP reaches them, the proposed plan is valid. If, on the other hand, one or more anomalies remain valid by the time the AP reaches them, their subgraphs are removed from the working graphmap and the above procedure is repeated until a valid path is found.

There can be cases where it might be convenient to wait for the expiration of an anomaly rather than taking a detour. For example, an anomaly may expire after one second, while the alternative route forces the AP to extend her journey by several seconds. To account for this we propose to implement an heuristic with a customisable cost threshold based on the AP's profile. If the cost of the alternative path is higher than the threshold, the AP is recommended to wait. If not, the detour is suggested.

4.6. Time-dependent shortest paths

Our *long term planner* intelligently avoids looping paths by regularly updating heat maps and assigning persistence times to anomalies. In this way the planner never returns to permanent obstacles, but may take advantage of crowding and obstacles that clear. Our current approach with heat maps assumes that crowding averaged over a period of time in the immediate past is a good indicator of average crowding for the same time period in the immediate future. This is reliable for short term predictions, but is less so over the longer term because long term averages may mask large peaks of crowding. With regard to anomalies, our planning algorithm takes a cautious approach, assuming that an active anomaly encountered in one proposed path should not be considered in future plans to the same point of interest.

Algorithm 1 describes the basis of our shortest path algorithm that considers timed anomalies, heat maps and user constraints. The algorithm finds the shortest path between the AP's current position and the closest point of interest. If points of interest are required to be visited in a specific order, it is assumed that the set *Targets* contains only those nodes corresponding to the next point of interest to visit.

The algorithm makes use of several functions. $K(e)$, $W(e)$ and $H(e)$ are as in (1). Function Edges(n) returns the set of outgoing edges of node n. Function Dest(e) returns the destination node of edge e. Function Anomalytime(e) returns the absolute time at which edge e will be available. This function returns 0 for all edges that are not part of an anomaly. Three functions are updated during the planning process. Function Dist(n) returns the currently known shortest distance to node n. This is initially ∞ for all nodes except the initial node, for which the function returns 0. Function Pred(n) returns the predecessor of node n, i.e., the node whose outgoing edge directly connects to n and gives rise to Dist(n). The function initially returns *null* for all nodes. Function Time(n) returns the estimated time to reach node n given the AP's average speed (denoted *speed*). The function initially returns ∞ for all nodes except the initial node, for which it returns 0.

Algorithm 1 Shortest path considering anomalies, heat and constraints

The initial node is the closest node to the AP
speed: the AP's average speed
Targets: a set of target nodes corresponding to the AP's points of interest
Visited: a set of visited nodes, initially containing the initial node
Unvisited: a set of unvisited nodes, initially containing all nodes except the initial node
$current \leftarrow$ initial node
while $current \notin Targets$ **do**
  for all $e \in \text{Edges}(current)$ **do**
    if $\text{Anomalytime}(e) > \text{Time}(current) \vee$
      $\text{Dest}(e) \in Visited$ **then**
      continue
    end if
    $d \leftarrow \text{Dist}(current) + K(e)W(e)/(1-H(e))$
    if $d \leq \text{Dist}(\text{Dest}(e))$ **then**
      $\text{Time}(\text{Dest}(e)) \leftarrow \text{Time}(current)$
        $+W(e)/(1-H(e))/speed$
      $\text{Dist}(\text{Dest}(e)) \leftarrow d$
      $\text{Pred}(\text{Dest}(e)) \leftarrow current$
    end if
  end for
  $Visited \leftarrow Visited \cup \{current\}$
  $Unvisited \leftarrow Unvisited \backslash \{current\}$
  if $|Unvisited| > 0$ **then**
    $current \leftarrow n \in Unvisited :$
      $\text{Dist}(n) \leq \text{Dist}(n'), \forall n' \in Unvisited$
  else
    report no possible path and quit
  end if
end while
Output the shortest path: backtrack from *current* to the initial node using Pred to identify predecessors

In trying to satisfy the conflicting constraints of dynamic motion planning in complex human environments we have considered many alternatives and refinements to our algorithms. There is no off-the-shelf perfect solution, given the inherent uncertainties and variability of the problem. In particular, finding time-dependent shortest paths is known to be hard and is itself the subject of active research [12,9,10,16]. Our present approach is a satisfactory compromise between efficiency and efficacy. We can imagine circumstances under which it might be challenged, but we propose to allow further development to be led by problems encountered in real applications.

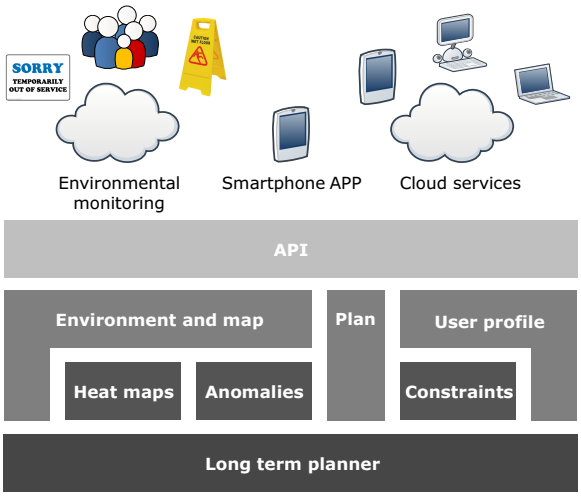


Fig. 3. Structure of the API. The layers are of increasing abstraction, where the public interface is flexible and extensible at runtime by the third-party services. The overall low complexity enables a broad choice of implementations, from a service in the cloud to a standalone smartphone app.

5. Implementation aspects

To develop our approach we have implemented two tools; a map designer and a simulator. The map designer is written in MATLAB and enables the AP to draw, load and save floor plans, as well as performing quad tree decomposition and graph construction. The user is provided with a GUI to freely draw geometric shapes (Figure 4) and generate the corresponding graph (Figure 5) to be used in the simulator.

The simulator is written in MATLAB and Java and allows the user to visually configure global constraints, heat maps, anomalies and all parameters required by the *long term planner*. To judge performance in a final product, the planning algorithm has been developed in Java and communicates with MATLAB through the integrated Java interface.

The algorithm presented in this work has been designed keeping flexibility in mind. We devised an API that abstracts the low level structures and exposes a simple but efficient interface. It is divided into a number of layers depicted in Figure 3. The bottom layer is represented by the *long term planner* itself, which is linked with the top level (the API) via three main blocks.

The first block is denoted "Environment and map" and, as the name suggests, allows external services to access and update information about the environment. Such data includes the map of static obstacles and walls, the heat maps and the anomalies. The lat-

ter can be grouped into categories, two being available by default: “wet floor” and “destination out of order”. New categories can be added at runtime upon request by the third-party services.

The second block, “Plan”, exposes the planning capabilities. Given the starting position, it is possible to query for the construction of the optimal path directed to one or more goals. The planner automatically considers the current status of the environment and biases the resulting trajectory according to the AP’s preferences. Moreover, alternative sub-optimal paths can be generated upon request, for example when the chosen path is blocked by an unforeseen obstacle detected by the short term planner.

The last block is the “User profile” and encapsulates the interface for accessing the global constraints and other user information, such as her location and the tuning parameters for dealing with anomalies and crowded areas.

This API can be installed and accessed practically anywhere, thanks to the low computational burden highlighted in Section 7.3. For example, it can be packaged in a standalone mobile application for providing the APs an interactive map of a shopping mall, or implemented as a cloud service, as described in the next section.

5.1. Case Study: a cloud service

In our case study, presented in detail in Section 8, we implemented the planner as a service in the cloud. The interface was written in C++, while efficient Java was used for the planning part. The standard Java Native Interface (JNI) provides the link between these elements. We anticipate that a production version will be entirely implemented in Java, to allow it to be deployed on a standard portable device under the Android operating system.

The map of the environment is stored using SpatiaLite[1], a lightweight serverless spatial database that allows performing queries in geometric space. A quad tree decomposition is then performed on the map and the resulting graph is used by the planner.

The communication with the remote clients takes place through exchange of JSON messages over a TCP link, in a request-reply mechanism, where the planner acts as a server.

[1] http://www.gaia-gis.it/gaia-sins/

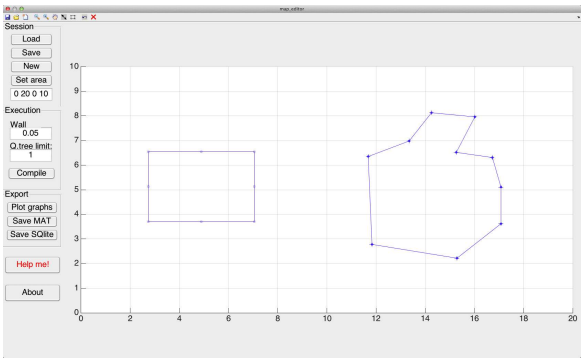


Fig. 4. Screenshot of the map designer tool showing an example floor plan. Enables the user to create maps and generate the associated graph, compliant with the *long term planner*.

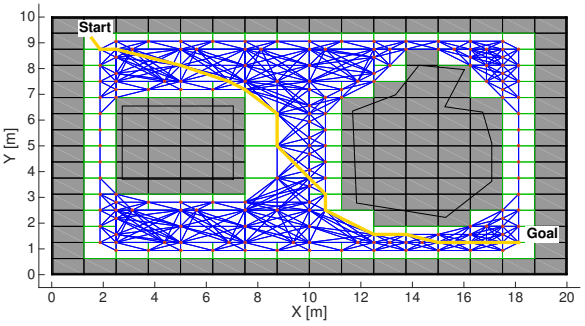


Fig. 5. Graph and a sample path generated from the map depicted in Figure 4.

6. Qualitative analysis

The chosen floor plan for the validation is a large room of approximately 200 m^2 with two non-aligned central columns. The starting point is set at the left hand side of the map, midway along the shortest wall. The goal is set at the opposite side of the room, such that the shortest path connecting the starting point to the goal is a horizontal straight line.

In the remainder of this section we will go through each feature separately and, finally, show a more complex simulation combining different features.

6.1. Global constraints

We show how the planner is able to deal with the user preferences when computing the plan. In the first simulation we put an undesirable zone in the middle of the room, overlapping the shortest path. In the second simulation, we instead identified a desirable zone (e.g., a restroom) close to the top wall of the map without interfering with the shortest path. In both simulations

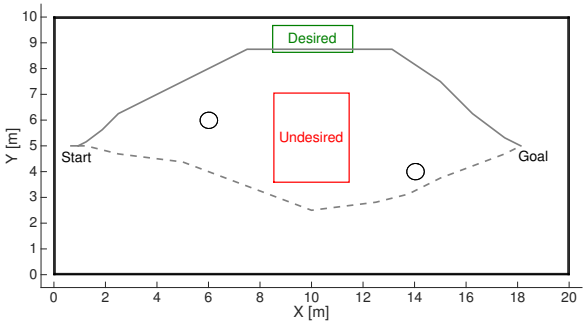


Fig. 6. Simulation with constraints. The picture shows two independent simulations of how the planner deals with desirable and undesirable zones. The continuous line is the result of the constraint "stay close to the desirable zone" and "stay away from the undesirable zone", while the dashed line addresses only the latter.

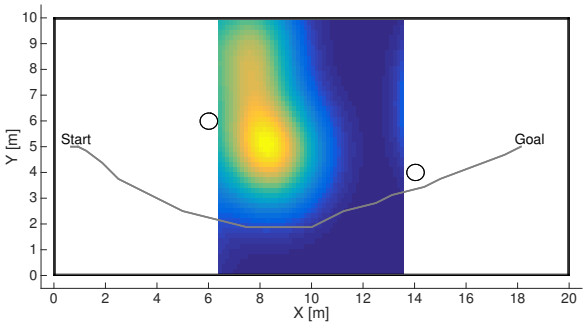


Fig. 7. Simulation with a heat map. The path computed by the planner is represented by the continuous line and bypasses the crowded region in the middle of the map (i.e., yellow area). The preference is for cold zones (i.e., blue-coloured areas).

the *radius* of the constraints is set to 1.5 m and the *intensity* is set to 2.

We ran these two simulations separately and the results can be seen in Figure 6. The planner correctly takes the constraints into account by properly bending and extending the original shortest path.

Should the undesirable zone be the only possible access point for reaching the goal, the planner can violate the constraints as long as the intensity is not $-\infty$ (i.e., *never* touch the undesirable zone).

6.2. Heat maps

We placed one rectangular shaped heat map in the centre of the room, covering the whole space between the two columns and the walls at the top and bottom of the figure. The planner is thus forced to go through the area covered by the heat map to reach the goal. We ran 50 simulations with different heat distribution generated by a sum of bivariate Normal probability density functions (normalised to the range $[0, 1]$) with random parameters. In all cases the planner correctly took into account the presence of the heat map. The outcome of one particular simulation can be seen in Figure 7, where the planner properly avoids hot (yellow-coloured) zones.

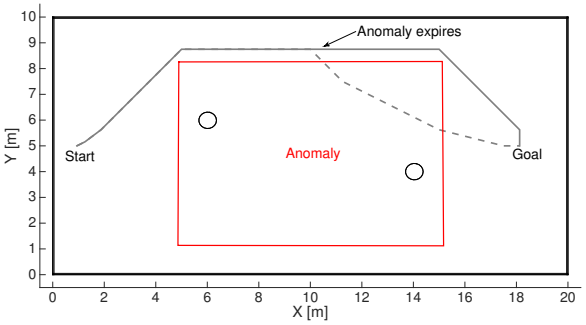


Fig. 8. Simulation with anomalies: two independent simulations are shown. The dashed line represents the path generated when the anomaly is set to expire half way to the goal. The continuous line, instead, shows the resulting path when the anomaly does not expire.

6.3. Anomalies

To test the handling of time based anomalies we set the average user speed to 0.5 m/s and we placed a rectangular anomaly in the middle of the room. Figure 8 depicts two paths constructed by the *long term planner* during two independent simulations with different durations of the anomaly. In this way we are able to show how the planner manages the disappearance of an anomaly. In the first run (dashed line) we configured the expiration of the anomaly in such a way that it expires when the user has covered approximately half of the path. In the second run (continuous line) the anomaly disappears after the user reaches the goal position. It is clearly visible that, in the first case, as soon as the anomaly expires the planner re-routes the user towards the shortest path, overlapping what was the area occupied by the anomaly.

6.4. Combination of features

The last validation test considers a combination of multiple features in one simulation. We placed one undesired zone, one desired zone, one anomaly and one heat map as shown in Figure 9. In particular, the undesired zone completely blocks the passage for reaching the goal. However, as visible in the previous figure, the *long term planner* is able to ignore the unfeasible con-

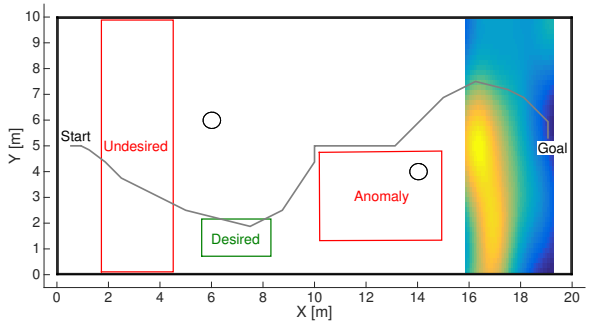


Fig. 9. Simulation with multiple features. The planner satisfies all the AP requests, but is forced to ignore the constraint for the undesired zone, as it is the only way for reaching the goal.

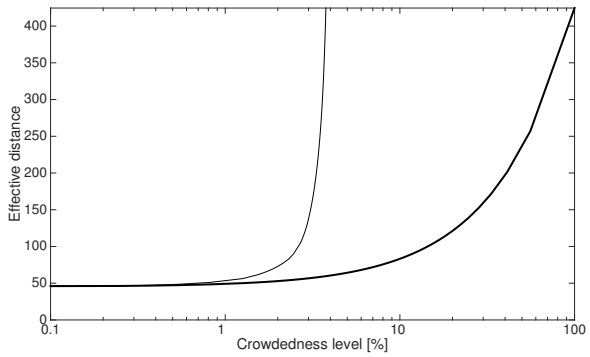


Fig. 10. Relation between level of crowdedness and effective length of the path. When the planner is aware of the heat maps in the environment, the *long term planner* is able to avoid the heat and the effective distance increases slowly with increasing crowdedness (thick line). Without this information, the *long term planner* just chooses the shortest Euclidean path, whose effective length increases exponentially.

straint. The path then bends towards the desired zone, bypasses the unexpired anomaly and, finally, avoids the crowded region represented by the heat map.

7. Quantitative analysis

We now go through the results of some simulations providing a quantitative analysis of the performance of the *long term planner*. The aim is to show that the benefits of using the *long term planner* are evident not only from a qualitative point of view, as shown in Section 6, but also from a well-defined set of performance metrics.

7.1. Heat maps

We demonstrate that the *long term planner* is able to provide better (i.e., quicker) trajectories when it is aware of the crowdedness in the environment.

We set up a simulation similar to the one in Section 6.2, where the heat map covers the environment as in Figure 7. We then iteratively increase the heat surface, simulating an expanding crowd, starting from no crowd (0% crowdedness) up to a completely crowded area (100% crowdedness). At each iteration we call the *long term planner* and we compute both the optimal path (e.g., considering the heat encoded in the effective distance) and the Euclidean shortest path (e.g., a straight line directed to the goal that passes through the crowded area).

The Euclidean shortest path $N_e = \{n_i \in N'\}_{i=i}^{k}$ is constructed by assuming $H(e) = 0$ in (1). The true cost W_e of N_e is then computed by removing the $H(e) = 0$ assumption, thus

$$W_e = \sum W'(e), \forall e = (n, n') \in N_e$$

The results are shown in Figure 10. The very slow growth of the effective length of the path considering heat (thick line) is clearly visible. The planner diverts the path to avoid the hot areas until this becomes impossible (i.e., when crowdedness reaches 100%). In contrast, the effective length of the Euclidean shortest path explodes exponentially (thin line), making the planner unable to find a path when the average crowdedness level is greater than 5%.

7.2. Global constraints

The simulations presented in this section show how the planner interprets the *intensity* parameter of an undesired or a desired global constraint. The environment and the position of the desired/undesired locations are the same as those considered in Section 6.1 and illustrated in Figure 6. We identified this particular scenario because it is a worst case situation: the desirable *location* is at the farthest possible distance from the shortest path and the undesirable *location* conflicts with the shortest path.

For simplicity and without any loss of generality, in these simulations we define $\tilde{K}$ as a linear function that is monotonically increasing for desired constraints, and monotonically decreasing for the undesired ones.

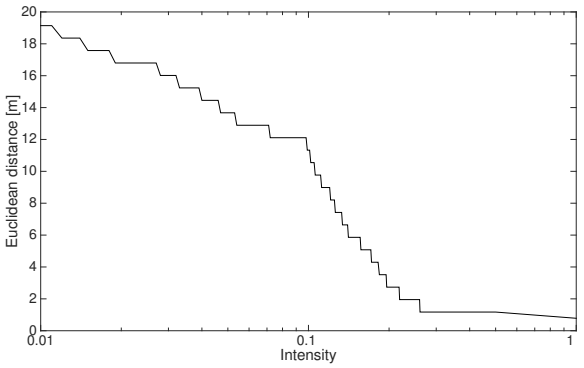


Fig. 11. The effect of *intensity* on the minimum Euclidean distance from a path to a desirable zone on a particular simulation run. As *intensity* increases the path is attracted towards the desirable location: the Euclidean distance decreases.

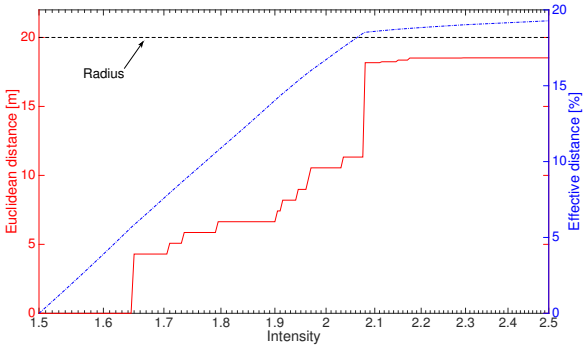


Fig. 12. The effect of *intensity* on the minimum Euclidean distance from a path to an undesirable zone on a particular simulation run. As the *intensity* increases the path is pushed away from the undesirable location (the Euclidean distance increases) until it approximates the specified *radius* (20 m, dashed line). The constraint is actually implemented with respect to the effective length of the path, which is shown for comparison.

To measure the characteristics of a constraint for a desirable zone, we set the *location* as far as possible from the Euclidean shortest path and we fixed the *radius* to a small value. We then iteratively executed the planner with increasing *intensity* and we computed the minimum direct Euclidean distance of the path from the *location* (i.e., not considering the graph). The results are reported in Figure 11. As expected, the minimum Euclidean distance between the path and the desirable zone decreases as *intensity* increases. The steps in the plot are due to the quantisation of the free space imposed by the underlying graph.

A similar procedure was carried out using a constraint for an undesirable zone. We set the location of the constraint midway along the Euclidean shortest path and fixed the *radius* of the constraint to be the largest possible value (in the simulations the limit is the distance from farthest wall). The *intensity* of the constraint was then iteratively increased and we computed the minimum direct Euclidean distance of the resulting path from the *location*. The results are shown in Figure 12. We observe that as the *intensity* grows, the planner "pushes" away the constructed path until the minimum Euclidean distance is close to the *radius*. Again, the steps in the plot are due to quantisation of the free space.

The relations highlighted in these paragraphs are strictly dependent on the considered environment. Different locations, position of obstacles or constraints lead to different relations. An open problem, to be addressed in future work, is how to generalise the relationship between these parameters.

7.3. Computing time

We tested the performance of the *long term planner* on the BeagleBoard xM[2], an affordable embedded board equipped with an ARM processor running at 1 GHz with 512 MB LPDDR RAM. The operating system is Ubuntu 12.04 and the Oracle Java Virtual Machine 1.8.0_u6 is installed.

Our goal was to verify the feasibility of an online implementation in a realistic scenario and the scalability of the performance with increasing dimensions of the graph. We thus designed a map of a large shopping mall (500 m x 250 m) and performed quad trees decomposition (Section 4.1) with different minimum quadrant resolutions, varying from 4 m to 0.8 m. This way the resulting graphs had different sizes, from 1686 nodes and 13832 edges, to 23016 nodes and 264026 edges.

For each graph we prepared a benchmark script that sets up the Java planning algorithm and queries 20 times for a path between the same two points at the opposite sides of the shopping mall. We then timed both the setup phase (e.g., loading the graph structure in the planning algorithm) and each of the planning queries. Finally, we computed the mean (μ) and standard deviation (σ) of the timings.

The results are reported in Table 1. The worst case, as expected, occurs with the largest graph. In this case we measured $\mu = 1983$ ms and $\sigma = 239$ ms for the setup phase, and $\mu = 812$ ms and $\sigma = 139$ ms for the query phase. These results are encouraging and show

[2] http://beagleboard.org

Table 1
Performance of the *long term planner* on a BeagleBoard xM with different graph dimensions for both setup and query phase. Mean (μ) and standard deviation (σ) are reported for each phase.

Minimum	Graph size		Setup [ms]		Query [ms]	
quad tree cell [m]	Nodes	Edges	μ	σ	μ	σ
4.0	1686	13832	78	35	56	61
2.0	4404	43720	282	141	163	117
1.0	10133	108818	829	354	361	176
0.8	23016	264026	1983	239	812	139

that the current implementation, which we believe can easily be improved, is already reasonably fast for an online execution. Finally, it should be noted that in real scenarios the setup phase needs to be executed only when the graph structure (i.e., the floor plan) changes permanently.

8. Case study: the DALi project

Motivated by the same considerations presented in this paper, the DALi project[3] aims to devise the c-Walker, an intelligent "walker" (an assistive wheeled device) that detects the presence of other pedestrians in the environment, anticipates their intent and plans an appropriate path that is suggested to the user via a combination of audio, visual and haptic interfaces.

The motion planning algorithm is part of the so-called "Cognitive Engine" and follows the diagram depicted in Figure 1. In this particular case, the *long term planner* is the algorithm proposed in this work while the *short term planner* outputs a suggested trajectory and is reactive to the potentially uncooperative response of the AP [7].

8.1. Experiments

In October 2014 we ran an experimental campaign that involved several elderly people at our facilities. The goal was to test the functionalities of the walker as well as of the motion planning algorithm. To this end, we created a simulated shopping mall environment and recruited a cohort of 12 senior users. We asked each participant to choose a destination in the environment (Figure 13(a)) and then to follow the guidance suggestions of the walker.

[3] http://www.ict-dali.eu

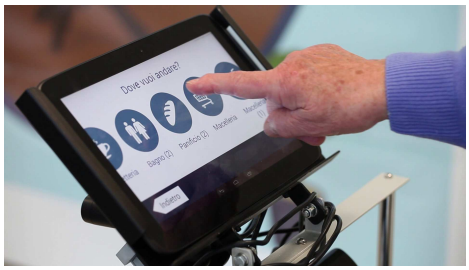
At the end of each test we collected results on the participant's performance and asked her/him to answer some questions about the quality of the guidance suggestion and personal satisfaction.

In addition, we selected a group of caregivers working in protected residences and proposed to each of them a tour through the functionalities of the system, where each of them could define hard and soft constraints and test the system. During each test we randomly triggered anomalies (Figures 13(b) and 13(c)) and heat maps (Figure 13(d)) to show the reactions of the system to such conditions. At the end of the "road show", we collected informal opinions and suggestions.

The impression we derived from reading the questionnaires collected from the APs and from talking to the care givers was of a general interest and appreciation toward the system and its functionalities (including the *long term planner*). Most users are keen on being actively engaged with future development activities. This motivates us in pursuing this line of research in the upcoming years.

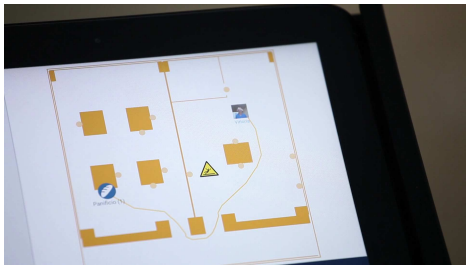
9. Conclusions

In this work we have presented an algorithm for long term motion planning in crowded public spaces. The algorithm applies to robotic platforms assisting the navigation of senior users in large and complex spaces. Key features of the algorithm are: 1. the ability to encode preferences in the user's profile on areas that should be avoided during the navigation and others that should be travelled across, 2. the consideration of time-dependent anomalies in making the right choice for a path, 3. the inclusion of crowdedness as a key parameter to take into account when estimating the time to complete a path. Our idea is to use quadtrees to generate a graph structure describing the space and en-

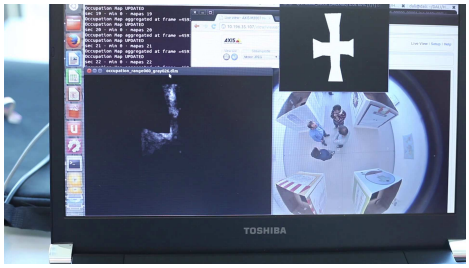


(a) Tablet interface for the *long term planner*

(b) Wet floor sign



(c) The *long term planner* reacts to the wet floor sign



(d) Heat map

Fig. 13. Pictures from the DALi project experimental campaign.

code user preferences, anomalies and heatmaps in the weight of the edges. We propose a modified version of the Dijkstra algorithm to identify the optimal path accounting for the time dependencies of the graph.

Our algorithm has been implemented as a cloud service that operates alongside a module for reactive (short term) planning and motion control, which are typically hosted on the robotic platform. Thanks to its flexible API and its low computational burden, the algorithm can be easily implemented in different ways, giving to the system integrators plenty of possibilities.

The different functionalities of the system have been validated in two ways. We have tested it through simulation scenarios and prepared a mockup simulating a realistic case study where the system was tested by a group of users and showcased to a group of caregivers.

This case study helped us to identify some borderline scenarios that require further analysis, especially when dealing with combinations of constraints. For example, when the user requires a "timed" constraint (e.g., "keep a toilet within 5 minutes walking distance"), when several constraints for desirable and undesirable zones appear to be placed one after the other, or when two or more constraints for desirable zones are placed at opposite ends of an environment. Simulations have shown that combinations of contrasting requests can be managed efficiently, even though an extensive analysis of this behaviour has not been carried out in the field. Nonetheless, the simplicity and the robustness of the proposed solution is very promising for an efficient handling of such complex situations.

Our future plans include lifting the planning algorithm to a social dimension, with motion plans organised for groups of people supported by a robotic platform, supporting constraints and anomalies specified in a probabilistic framework.

References

[1] P. R. A. Baker, D. P. Francis, J. Soares, A. L. Weightman and C. Foster. Community wide interventions for increasing physical activity. In: *Cochrane Database of Systematic Reviews*, **9**(1).

[2] P. Bhattacharya and M. Gavrilova. Voronoi diagram in optimal path planning. In: *International Symposium on Voronoi Diagrams in Science and Engineering*. IEEE, 38–47.

[3] J. T. Cacioppo and L. C. Hawkley. Perceived social isolation and cognition. In: *Trends in Cognitive Sciences*, **13**(10), (2009), 447–454.

[4] T. Carlson and Y. Demiris. Collaborative Control for a Robotic Wheelchair: Evaluation of Performance, Attention, and Workload. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, **42**(3), (2012), 876–888.

[5] D. Chen, R. Szczerba and J. J. Uhran. A framed-quadtree approach for determining Euclidean shortest paths in a 2-D environment. In: *IEEE Transactions on Robotics and Automation*, **13**(5), (1997), 668–681.

[6] A. Coles and A. Coles. LPRPG-P: Relaxed Plan Heuristics for Planning with Preferences. In: *International Conference on Automated Planning and Scheduling*. Association for the Advancement of Artificial Intelligence, 26 – 33.

[7] A. Colombo, D. Fontanelli, A. Legay, L. Palopoli and S. Sedwards. Motion planning in crowds using statistical model

checking to enhance the social force model. In: *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. IEEE, 3602–3608.

[8] F. A. Cosío and M. P. Castañeda. Autonomous robot navigation using adaptive potential fields. In: *Mathematical and Computer Modelling*, **40**(9-10), (2004), 1141–1156.

[9] D. Delling and D. Wagner. Time-dependent route planning. In: R. K. Ahuja, R. H. Möhring and C. D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg (2009). 207–230.

[10] U. Demiryurek, F. Banaei-Kashani and C. Shahabi. A case for time-dependent shortest path computation in spatial networks. In: *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10. ACM, New York, NY, USA, 474–477.

[11] E. Dijkstra. A note on two problems in connexion with graphs. In: *Numerische Mathematik*, **1**(1), (1959), 269–271.

[12] B. Ding, J. X. Yu and L. Qin. Finding time-dependent shortest paths over large graphs. In: *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '08. ACM, New York, NY, USA, 205–216.

[13] C. R. Dyer. The space efficiency of quadtrees. In: *Computer Graphics and Image Processing*, **19**(4), (1982), 335 – 348.

[14] R. Finkel and J. Bentley. Quad trees a data structure for retrieval on composite keys. In: *Acta Informatica*, **4**(1), (1974), 1–9.

[15] D. Fontanelli, A. Giannitrapani, L. Palopoli and D. Prattichizzo. Unicycle steering by brakes: A passive guidance support for an assistive cart. In: *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. 2275–2280.

[16] L. Foschini, J. Hershberger and S. Suri. On the complexity of time-dependent shortest paths. In: *Algorithmica*, **68**(4), (2014), 1075–1097.

[17] S. Gulati, C. Jhurani, B. Kuipers and R. Longoria. A framework for planning comfortable and customizable motion of an assistive mobile robot. In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. 4253–4260.

[18] D. Guo, C. Wang and X. Wang. A hierarchical pedestrians motion planning model for heterogeneous crowds simulation. In: *IEEE International Conference on Information and Automation, 2009*. IEEE, 1363–1367.

[19] P. Hart, N. Nilsson and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. In: *Systems Science and Cybernetics, IEEE Transactions on*, **4**(2), (1968), 100–107.

[20] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. In: *The International Journal of Robotics Research*, **30**(7), (2011), 846–894.

[21] L. Kavraki, P. Svestka, J.-C. Latombe and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In: *Robotics and Automation, IEEE Transactions on*, **12**(4), (1996), 566–580.

[22] T. Kruse, A. K. Pandey, R. Alami and A. Kirsch. Human-aware robot navigation: A survey. In: *Robotics and Autonomous Systems*, **61**(12), (2013), 1726 – 1743.

[23] M. Lahijanian, S. Almagor, D. Fried, L. E. Kavraki and M. Y. Vardi. This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence, 3664–3671.

[24] Y. Lasovsky and L. Joskowicz. Motion planning in crowded planar environments. In: *Robotica*, **17**, (1999), 365–371.

[25] S. LaValle and R. Sharma. Robot motion planning in a changing, partially predictable environment. In: *IEEE International Symposium on Intelligent Control, 1994*. IEEE, 261–266.

[26] S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Iowa State university (1998).

[27] S. M. LaValle. *Planning Algorithms*. Cambridge University Press (2006).

[28] J. Miura and Y. Shirai. Hierarchical vision-motion planning with uncertainty: Local path planning and global route selection. In: *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on*, volume 3. 1847–1854.

[29] Y. Morales, N. Kallakuri, K. Shinozawa, T. Miyashita and N. Hagita. Human-comfortable navigation for an autonomous robotic wheelchair. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. 2737–2743.

[30] C. H. Papadimitriou. The euclidean travelling salesman problem is np-complete. In: *Theoretical Computer Science*, **4**(3), (1977), 237 – 244.

[31] L. Rizzon and R. Passerone. Embedded soundscape rendering for the visually impaired. In: *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*. 101–104.

[32] S. Scheggi, M. Aggravi, F. Morbidi and D. Prattichizzo. Cooperative human-robot haptic navigation. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. 2693–2698.

[33] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte and D. Schulz. Minerva: a second-generation museum tour-guide robot. In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 3. 1999–2005 vol.3.

[34] J. Tumova, L. Castro, S. Karaman, E. Frazzoli and D. Rus. Minimum-violation ltl planning with conflicting specifications. In: *American Control Conference (ACC), 2013*. 200–205.

[35] J. Van Cauwenberg, V. Van Holle, D. Simons, R. Deridder, P. Clarys, L. Goubert, J. Nasar, J. Salmon, I. De Bourdeaudhuij and B. Deforche. Environmental factors influencing older adults' walking for transportation: a study using walk-along interviews. In: *International Journal of Behavioral Nutrition and Physical Activity*, **9**(1), (2012), 1–11.

[36] C. Warren. Global path planning using artificial potential fields. In: *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*. 316–321 vol.1.

[37] H. A. Yeom, C. Keller and J. Fleury. Interventions for promoting mobility in community-dwelling older adults. In: *Journal of the American Academy of Nurse Practitioners*, **21**(2), (2009), 95–100.