Integrating Time Performance in Global Path Planning for Autonomous Mobile Robots

A. R. Diéguez, R. Sanz* and J. L. Fernández
Systems Engineering and Automation Dept., University of Vigo
Spain

1. Introduction

Global path planners for autonomous mobile robots have been demonstrated to map optimal routes in structured environments. The search for the optimum path from a given starting point to a destination is usually formulated as a graph search problem.
Over the years, much progress has been made in efficient methods for global motion planning in static environments. A number of both exact and approximate approaches in this field (see, for example, Latombe, 1991; Hwang & Ahuja, 1992; LaValle, 2006). However, real world scenarios are intrinsically dynamic due to the presence of non-modelled obstacles, both static and moving. Path planning in such environments is an area of active research. Different methods have also been proposed (Heero, 2006; Jaillet & Simeon, 2004). Some approaches use robot sensors to build a model of the environment incrementally even when the real world is static. In such circumstances, a new path must be calculated every time the model is updated. The online integration of this new information in a global map is usually complex and time consuming.
Path planning with dynamic objects can be addressed by using explicit time representation to convert the problem into an equivalent static one that can then be solved using an existing static planner. However, this increases the dimensionality of the representation and requires exact motion models for moving objects (Szczerba & Chen, 1995; Hsu et al., 2002). Note that, in many circumstances (such as the presence of people near the robot in a tour-guide autonomous application), it is not possible to explicitly model the unpredictable behaviour of humans and robot movements with a time-extended representation (Philipsen et al., 2007).
In contrast to previous works, our global path planning approach is able to take into account the effect of persistent delays caused by unpredictable objects (such as people in corridors or lobbies) by considering these delays for future path planner executions.

2. Problem statement

Depending on the nature of the problem, different optimality criteria can be used to minimize the overall cost of the proposed path. Although the most frequently used cost function is based on the branch length between two nodes, in many cases it is more attractive to minimize traveling time rather than the distance traveled by the robot. Our approach focuses on problems in which this metric is of interest.

Due to the presence of non-modeled obstacles (both static and dynamic) in the environment, the robot must be able to avoid unexpected obstacles by means of a reactive behavior. This causes deviations from the original trajectory and possibly delays accomplishment of planned tasks.

The research on enhancing global path planning described in this chapter is intended for mobile robots navigating in partially known indoor environments. The method is based on a graph approach that adapts graph weights by integrating traveling time measurements from real task executions when the robot repeatedly follows the same paths.

The technique uses periodic measurements of time and the positions reached by the robot while moving towards the goal to modify the costs of the branches (Diéguez et al., 2007). These weights are good options for branch costs while the robot moves in the presence of non-modeled obstacles. Consequently, when trying to minimize the time needed to move from any given point to any given destination in dynamic environments, the search for a time-optimal path in a static global map produces better quality results than the use of a distance metric.

Another important aspect of our approach is its independence of the existence of a local motion planner or reactive guidance modules in the robot navigation architecture. This is especially attractive because it makes it possible to incorporate the time delay caused by non-modeled obstacles even in a situation where obstacles cannot be included in the map, such as people walking in crowded corridors or in a waiting room. Likewise, it makes the costs of the branches more realistic because they represent the time needed to travel in the environment.

In simulations and real experiments, results are generally better than using only the information on the obstacles stored in the global map, while the computational cost is significantly lower.

In order to illustrate the above problem statement, Figure 1 shows a simple test carried out with a mobile robot system developed in our Lab (Diéguez et al., 1998). Two possible trajectories are available to arrive at point 4 from point 1. Initially, segments 1-3-4 are chosen by the path planning algorithm because this is the shortest path, but three static non-modelled obstacles (grey boxes) found in the environment cause an increase in traveling time. Broken lines represent the actual trajectories followed to avoid the obstacles. Table 1 summarizes the results of two experiments with the robot at a constant speed of 0.5 m/sec. These non-modeled obstacles caused an important increase in the time needed to follow segment 1-3. Our global path planner recognizes this kind of situation and, as a consequence, selects segments 1-2-3-4 in further path planner executions.

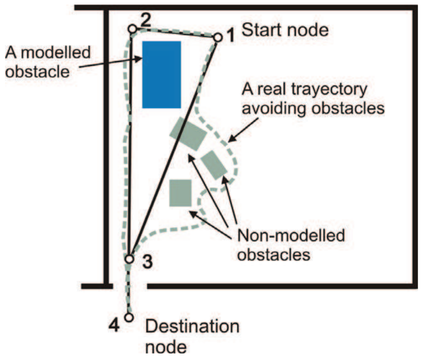


Figure 1. Mobile robot real trajectories in an experience with non-modelled obstacles

	Segment lengths	Estimated time	Actual time	Deviation
Segments 1-2-3-4	7.82 m.	15.64 sec.	16.31 sec.	4.28 %
Segments 1-3-4	6.11 m.	12.22 sec.	17.12 sec.	41.10 %

Table 1. Experimental time measurement in actual trajectories

We propose a different approach to dealing with global path planning in the presence of non-modelled obstacles that uses an aggregate time-cost function to find an optimal global path. The method uses traveling time and position measurements to calculate the time cost of each branch. We also propose a geometric method to modify the cost value, for each graph branch involved in the paths, to include the traveling time information.

Although our approach does not eliminate the need for a local motion planner in our mobile robot navigation architecture (Diéguez et al., 2003), it enables incorporation of the time delay caused by non-modelled obstacles even in the situation where these obstacles cannot be included in the map (waiting rooms or crowded corridors). It makes the cost of the branches more realistic because the time necessary to travel in the environment is represented. In many real experiments, results are generally better than using only the information on the obstacles stored in the global map, while the computational cost is significantly lower. For this reason, our method is especially attractive for incorporating the effects of mobile obstacles to the planning process.

The rest of the chapter is organized as follows: Section 3 presents the global path planning algorithm and briefly describes the skeleton (structure that represents the free space of the environment) building algorithm; the cost adaptation algorithm based on traveling time is outlined in Section 4; and some simulated results are given in Section 5. The evaluation of a real experiment with our mobile robot system is detailed in Section 6. Finally, some conclusions are given in Section 7.

3. Global path planning algorithm

The global path planning algorithm uses all currently available information about the static obstacles in a workspace to find the shortest path. In our approach, two different data structures were combined for this purpose: a cell map which containing prior knowledge of the workspace, and a path graph which is a topological representation of the free space. The cell map was created from a CAD map of the environment, whereas the path graph was created from the cell map using a wave front expansion algorithm generating a skeleton (Latombe, 1991). The skeleton is a connectivity network representing all possible routes in the free space and their intersections. The graph obtained this way allows two points anywhere in the workspace to be linked. The search for a feasible path is reduced to a graph search problem. In our method, both data structures were combined to accelerate the computation of the skeleton and to minimize search time for the optimum path.

The path resulting from a simple concatenation of the branches selected by the global path planner is called the *skeleton route* (see Table 2 for a summary of definitions). This trajectory may contain undesirable bends and detours that are unsuitable for the path. For this reason, once the skeleton route is obtained, a segment route is generated removing these imperfections and converting the route into a sequence of straight segments. Each segment is then linked to its adjacent one using smooth curves. This smoother version of the segment route is passed to the navigation module of the robot, in accordance with the model proposed in Diéguez et al. (1995).

	Definition
Skeleton	Any type of connectivity network representing the free space (e. g. a Voronoi diagram)
Node	Intersections in the skeleton
Branch	A portion of the skeleton between two intersections (nodes)
Optimum path	Fastest path between two points in the skeleton
Segment	A straight line composed of any (optimal) trajectory
Real trajectory	Route actually traveled by the robot
Segment threshold	A distance limit to a segment to compute control points
Control point	Point of the real trajectory close to a segment path

Table 2. Basic term definitions related to the global path planning algorithm

The segment route is used in our approach to adapt the weighting factors for each branch of the skeleton route. These costs are calculated from accumulated statistics of traveling times while the robot is moving through the workspace along different routes. These statistics were used to make the weights associated with each branch of the skeleton route closer to reality.

The method to calculate the optimum path consists of the following steps:

Step 1: Creating the cell map. An internal representation based on a cell map is constructed from a map of the environment stored in an external file.

Step 2: Building the skeleton of free space using an obstacle growth algorithm. A distance wave-front flows around the obstacles through all the free space in the environment. The skeleton is the set of cells where two or more wave-fronts meet (Latombe, 1991). The technique developed, unlike that used in similar algorithms, does not need to compute or store any values, making it possible to compute the skeleton rapidly and easily.

Step 3: Constructing a graph. This graph is the topological representation of the previously calculated skeleton. We have chosen this data structure because it is best suited to perform searches. A graph branch is a series of points linking two places in the free space. A graph node represents a point where two or more branches meet. Two nodes can be connected by one or more branches. A branch cost is initially defined as the estimated elapsed time for a robot to move from one node to another. The location of a node is represented by its coordinates. The graph is pruned and this reduction is transposed to the skeleton, thus eliminating irrelevant branches.

Step 4: Computing the optimum route using an exhaustive search algorithm on the graph. We implemented a breadth-first search strategy without repeating states (Russell & Norvig, 1995). Although in comparison with other non-informed methods this search strategy often requires a great amount of memory when working with cluttered environments, the memory problem does not arise in indoor structured environments due to the short number of generated states.

The path thus computed in the skeleton is called the *skeleton route*. This path is shortened by linking points of maximum visibility in the skeleton route. This is the path used to evaluate the traveling time function described in Section 4. The segments of the shortest path are then connected with arcs of circumferences and a velocity reference is assigned to each segment.

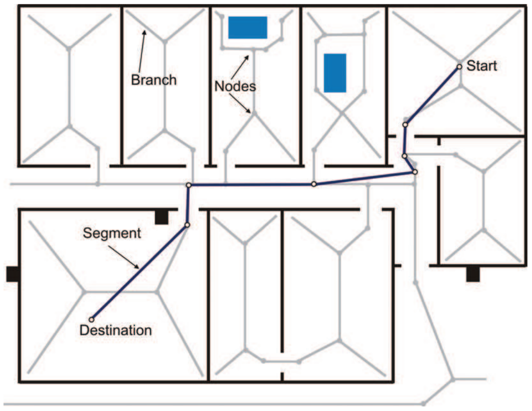


Figure 2. Workspace with rooms and corridors, with skeleton (in grey) and segment route (thin line)

Figure 2 shows a workspace describing the components used in our global path planner. The skeleton and the shortest path to arrive at the bottom right corner from the top left corner generated by the path planner are also depicted.

3.1 Description of the skeleton building algorithm

The algorithm developed to build the skeleton avoids the computation of potential field values and so dramatically reduces execution time. The algorithm iteratively grows all the obstacles (i.e. it thins the free space). The skeleton is the set of cells where these layers meet. This can be interpreted as a wave-front expansion, with each cell where two or more wave fronts meet set as a skeleton cell.

For the implementation of this algorithm, we only need the cell map. Each layer (wave front) is generated by labeling the neighboring cells of the previous layer with a temporary value. Thus, the layers use alternate cell values, it being necessary only two different temporary values. Thus, the range of values necessary can be represented by a single byte for each cell.

In order to generate each layer, a mask-based case identification method is used. This method functions on the basis of considering each cell and its eight immediate neighbors. The 3x3 resulting matrix is compared with the contents of a table with all possible cases to decide whether the central cell should be labeled as part of the layer or should be left empty. Only free (empty) cells are considered, so the 3x3 case can be represented by an 8-bit number (one bit for each cell surrounding the central one). This table only needs, therefore, to reflect 255 different cases.

When no more of the empty cells of the map can be labeled (that is, it cannot be incorporated as part of a layer), the algorithm terminates. The remaining empty cells constitute the skeleton. Figure 3 shows two examples where the central cell must remain empty and another two cases where the central cell should be labeled as part of the current growing layer.

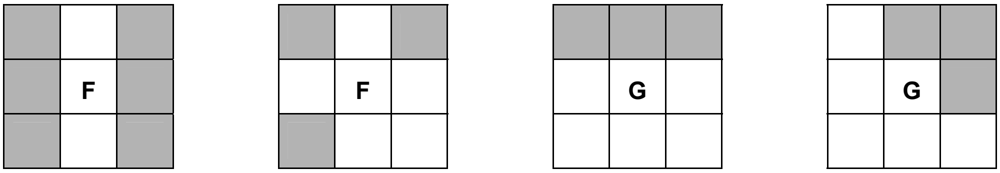


Figure 3. The first two cases must leave the central cell empty. In the other two situations, the central cell must be labeled as part of the current growing layer (G means "grow" and F means "free")

This algorithm is faster and uses less memory than the usual potential field-based wave front expansion methods. Moreover, the skeleton computed in such way is the most distant skeleton from the obstacles, leading to a lower probability of collision.

4. A geometric method for branch weighting factor calculation

This section describes the improvement incorporated in the global path planning algorithm in order to take into account statistical measurement of traveling time. The proposed method employs a geometric approach to modifying the weight of each branch according to the time spent by a robot navigating along the global path. The algorithm first maps each point of the actually performed route on the segment route. Each route segment is then associated with its corresponding branches and finally, the results are fused with the previous traveling time weights.

4.1. Mapping a real route on the segment route

Given a set of coordinate points actually performed by the mobile robot, a group of control points are selected. Control points are points common to both trajectories. Moreover, a coordinate point in the real route, which is close enough (a fixed distance threshold) to the segment path, can be regarded as a control point. Control points are employed as temporal references for traveling time assignment to the segments.

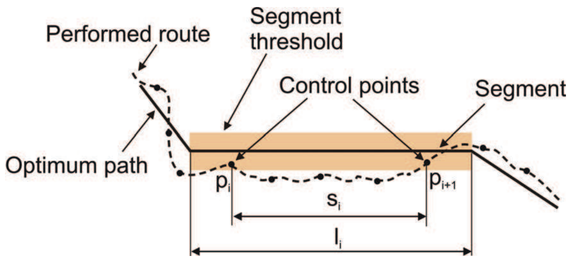


Figure 4. Mapping two control points in the same segment

Let us assume two consecutive control points (p_i, p_{i+1}) relating to the same segment whose length is l_j (Figure 4). The traveling time between the points is t_i, and s_i is a span over this segment, and it is assumed that their contribution to the traveling time of the segment is $t_i * s_i/l_j$. This means that the contribution to the segment traveling time is the time difference between both points weighted by the share of the segment spanned. Therefore, the final traveling time of a segment is the accumulation of all the contributions made by each pair of points related to the segment.

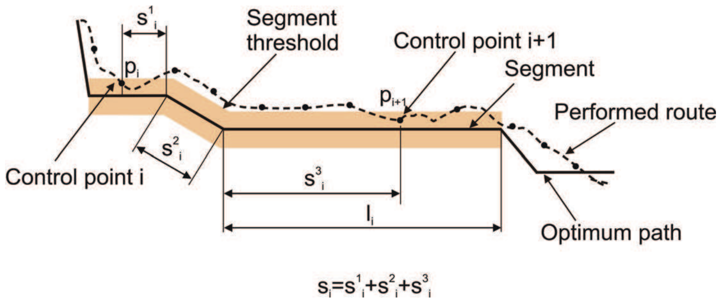


Figure 5. Mapping two control points in different segments

When a pair of consecutive control points spans more than one segment (or even fractions of segments), a similar procedure is implemented (Figure 5). Let us denote again by s_i the span over the segment route. Considering as l_j. the portion of the segment affected by the pair of points (note that l_j differs from the total length only for the first and the last segments), it is assumed, once more, that the contribution to the traveling time of the segment *j* is $t_i * s_i / l_j$. In the case of the segment being totally subtended by the pair of control points, l_j equals its total length.

Thus, once all the contributions of the control points in the segment have been established, the traveling time weight for a segment j is calculated as:

$$T_j = (1/ l_j) \cdot \Sigma_i\, t_i \cdot s_i \qquad (1)$$

where *i* ranges from *1* to the total number of control points in the segment.

To summarize, the procedure for this stage is as follows:

i. Find the corresponding segment for each control point and the related information (distance, etc).
ii. Remove the control points further than a threshold from its corresponding segment.
iii. Remove all the control points corresponding to the same end of the segment, except for the first and last control points.
iv. Remove intermediate control points until only two control points in each segment remain.
v. Compute and assign the traveling time weighting for each segment.

4.2. Mapping segment route on branch route

The aim of the second phase of the algorithm is to transfer the segment traveling time to the branches of the graph that make up the skeleton route.

We only consider the control points that are the ends of segments (except for the first and the last one) because they are common points for both routes. Let us denote by sj the span of the segment on the skeleton route. The velocity v_i for the segment *j* is defined as S_j/T_j ; and T_i is its traveling time weight. Thus, the traveling time weight for each branch is the accumulation of the weighted contributions of each segment related to this branch. Calling l_{ij} the portion of the branch *i* subtended by segment *j*, the increase of traveling time for the branch due to the segment is $v_j * l_{ij}$. The total time weight for a branch *j* is:

$$W_j = \Sigma_i\, T_i * S_i \qquad (2)$$

where *i* ranges from *1* to the total number of segments related to the branch *j*.

4.3. Fusion of costs

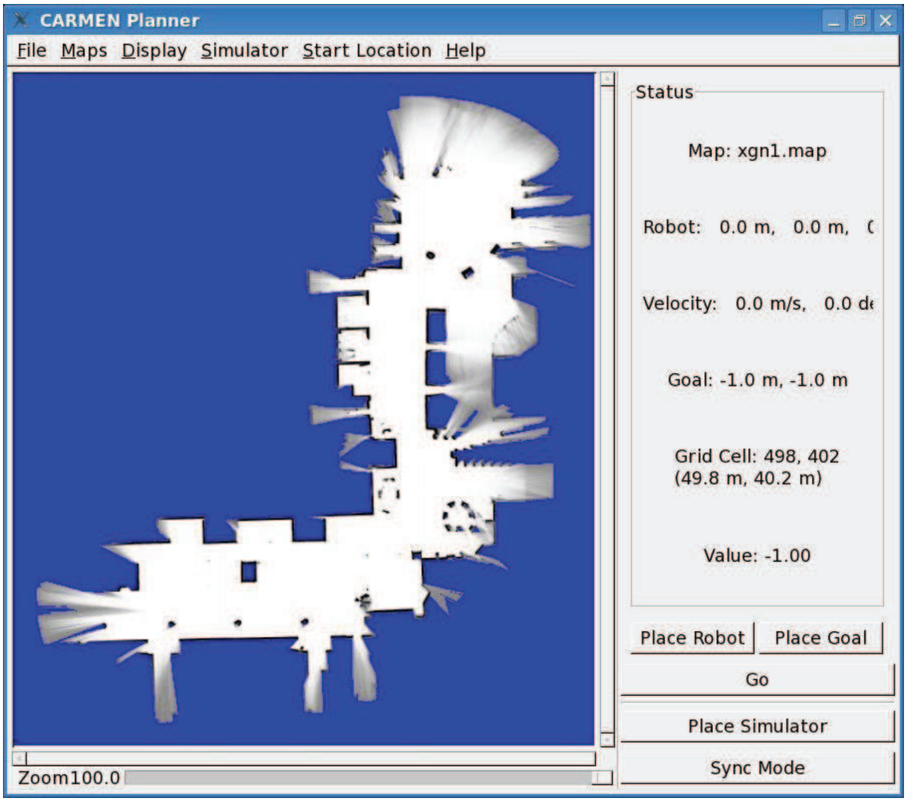
The weighting factors so far computed for each branch of the skeleton route are fused with the previously existing values using the following formula:

$$W_i = \alpha(c)\, W_0 + (1 - \alpha(c))\, W_{i-1} \tag{3}$$

The confidence measure (*c*) is a function of the confidences computed in the two previous phases. The factor *α(c)* has a value between *0* and *1*, and takes into account deviation of the new traveling time from the existing one. The cost of each branch is calculated as a weighted sum of the costs calculated so far and a constant that is a function of the length of the branch. This approach makes the final value more stable.

5. Simulation results

This section presents the behavior of the described technique in a series of simulations. Figure 6 depicts the simulation environment perceived by the sensors of our robot. This map corresponds to a real environment in the Santiago de Compostela Conference Center (Spain) (Fernández et al., 2008), with corridors, rooms, lobbies and stands.



Map: xgn1.map

Robot: 0.0 m, 0.0 m, (

Velocity: 0.0 m/s, 0.0 d

Goal: -1.0 m, -1.0 m

Grid Cell: 498, 402
(49.8 m, 40.2 m)

Value: -1.00

Figure 6. Environment used for the simulation

	Branch sequence	Skeleton route time	Estimated path time	Simulated path time	Deviation: estimated vs. simulated path time
Trajectory A	1→8→9→10→11→13	107.44	78.55	86.02	6.9%
Trajectory B	1→2→7→13	86.22	67.32	95.94	33.2%
Trajectory C	1→2→3→4→5→6→13	100.97	81.41	112.81	31.1%

Table 3. Estimated and simulated times (in seconds) for three alternative routes between the start and the goal

Figure 7 shows the initial position as a circle on the left, and the destination as a circle on the right. In order to reach the goal, the robot has three candidate routes, summarized in Table 3.

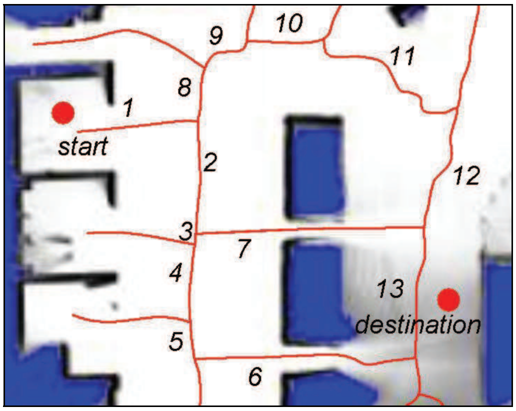


Figure 7. Simulation: skeleton of the environment, branch labels, and start and destination points for the route

	Skeleton time (sec.)	Simulated time (sec.)	Deviation: theoretical vs. simulated time
branch #1	17.73	16.73	5.6%
branch #2	16.30	11.94	26.8%
branch #7	33.02	51.99	-57.5%

Table 4. Weighting factor values for the simulated experiments

A series of tests simulating navigation by the robot in the presence of non-modeled obstacles was carried out by performing the three trajectories summarized in Table 3 and depicted in Figure 8. Each skeleton route was transformed into a segment route (straight segments) actually fed to the robot, as can be also seen in Figure 8. However branches 6 and 7 are located in narrow and busy corridors that slow down the robot run. This situation is numerically shown in Table 4. The delays are caused by both static and dynamic obstacles. The static obstacles make the robot step aside from the desired path, and the dynamic ones make the robot slow down or even stop until the obstacle (e.g. a person or another robot) has passed. In this case, there is no deviation from the segment route but, even so, a significant time delay is caused.

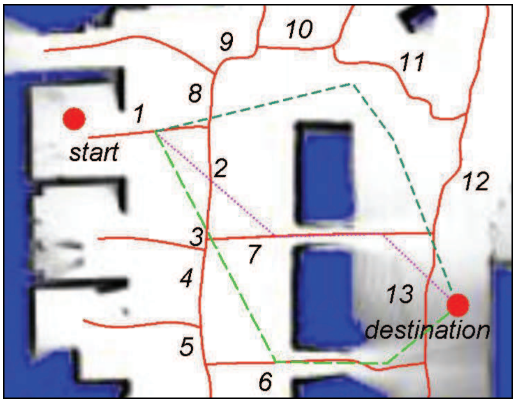


Figure 8. Skeleton and three possible segment routes

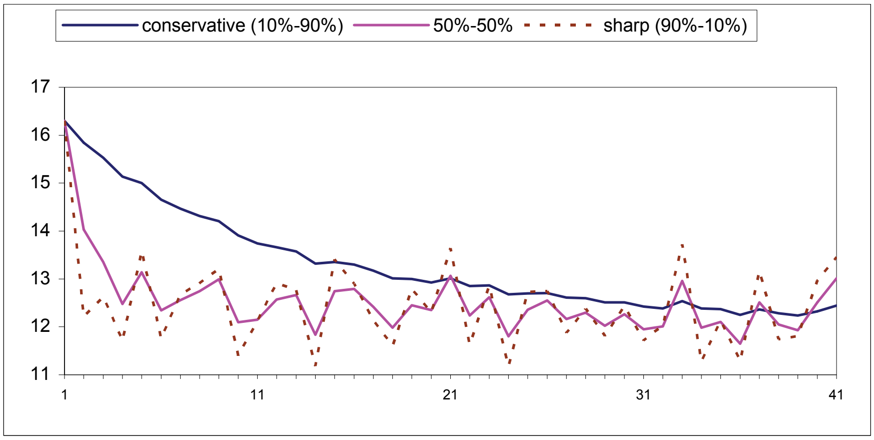


Figure 9. Branch weight convergence with different fusion ratios

After several simulations it becomes clear that the deviation from the initial cost differs depending on the branch under consideration (see Table 4). The initial weight is a function of the length of the branch. It must be pointed out that a constant speed along the whole trajectory was used in all the simulated experiments.

In order to study the relationship between fusion coefficient and branch weight convergence, several fusion ratios were used. The results can be seen in Figure 9. As expected, conservative fusion coefficients lead to slower and smoother convergence curves.

Based on this weight information, the global path planner updates the costs of the involved branches. Further executions of the global path planner choose trajectory A instead B or C, thus avoiding the aforementioned narrow, crowded corridors.

6. Experimental results with a real robot

In this section we summarize the results of a series of real-time experiments carried out with our mobile robot in the School of Engineering in the University of Vigo. The robot (called Rato) used in these experiments is based on a modified B21 RWI platform. It is equipped with a ring of 24 sonar proximity sensors and 24 bumpers on the enclosure connected with four CAN slave modules that are attached to the computer through a CAN-USB adapter card. The robot is also equipped with a SICK PLS 200 laser range finder. This sensor measures the distance to obstacles in the vicinity of the robot.

A layered software architecture is used, consisting of task scheduling, path planning, navigation, and obstacle avoidance components, each of which relies on the abstraction provided by the previous level. The robot architecture is implemented as a collection of asynchronous processes. IPC (Inter Process Communication), developed at Carnegie Mellon's Robotics Institute, is used to integrate the system and interprocess communication (Simmons, 1994). The global path planning method described here is included in this hierarchical navigation architecture (Figure 10). This navigation system combines a reactive part (BEAM module) with a deliberative one represented by the navigator and task planner modules. The advantages of both systems are thus obtained; a quick response to events that require it, and the effectiveness of the deliberative systems, as actions are regulated by references obtained by planning.

The tasks requested by the user are handled by the task planner that determines the places to visit and the order in which to do so. As long as there are tasks to be completed, this module is in charge of the selection of new destinations as the previous destinations are reached.

The goal point and the estimated position are used by the path planning module (Navigator) to determine how to travel efficiently from one location to another.

The reactive system will follow this direction avoiding any obstacles that appear. The information obtained from the sensors is integrated into a grid that represents the robot's environment. This grid is used by the localization module to estimate position.

Each module in Figure 10 is a Linux process that exchanges information with other modules using IPC, which provides a publication-subscription model. Each application registers with the the central server and specifies what types of messages it publishes and what types it listens for. Any message passed to the central server is immediately copied to all other subscribed processes.

The dynamic path planning algorithm outlined earlier is implemented as a single process.

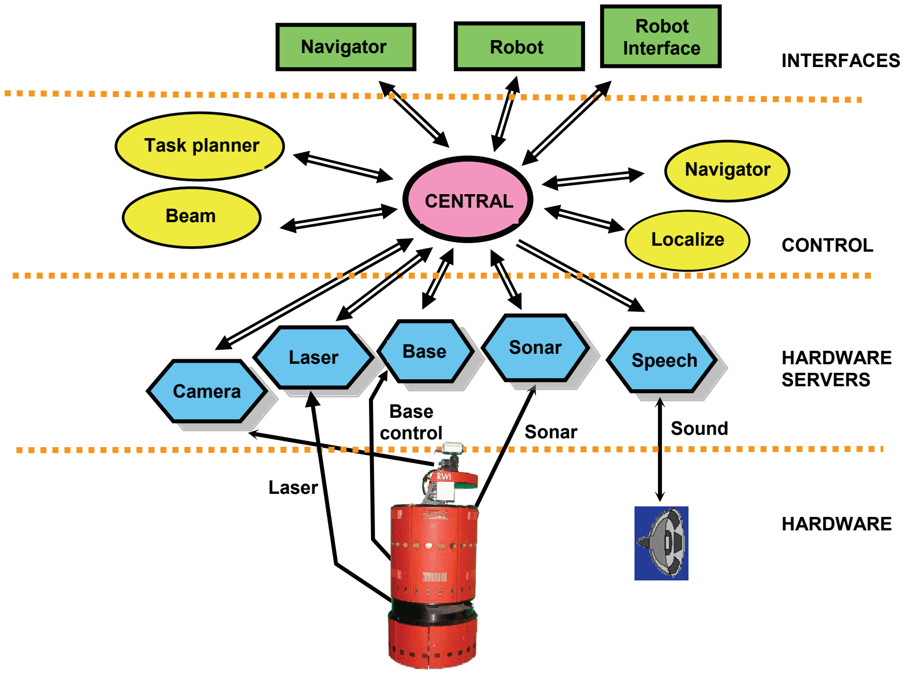
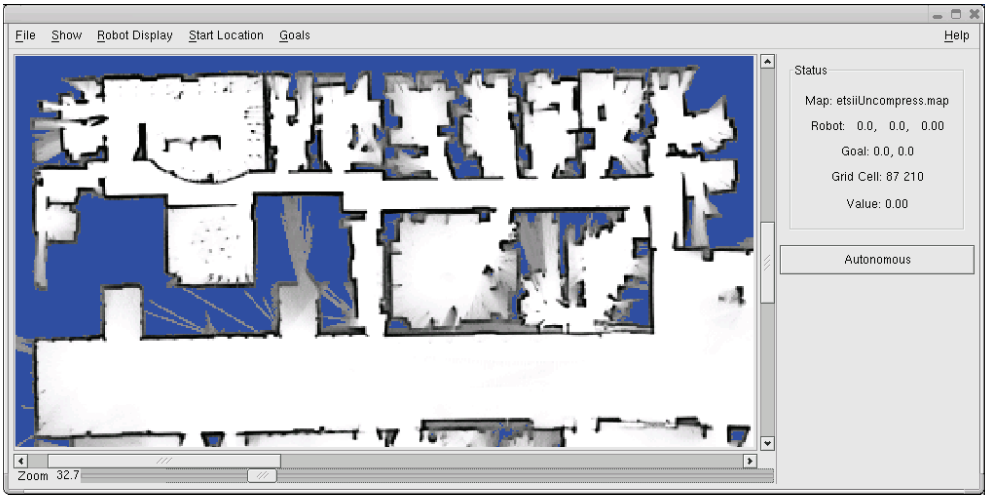


Figure 10. Graphical representation of the navigation architecture



Map: etsiiUncompress.map
Robot: 0.0, 0.0, 0.00
Goal: 0.0, 0.0
Grid Cell: 87 210
Value: 0.00

Figure 11. Environment of the experiment with the free space graph

Figure 11 shows the section of the environment (as perceived by the robot) which basically comprises offices, laboratories and corridors.

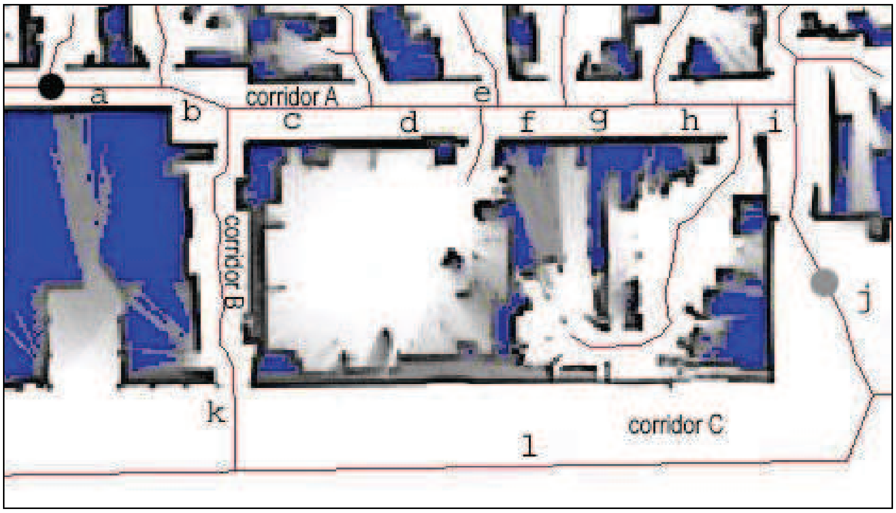


Figure 12. Branch labels, start point (black circle on the left) and destination (grey circle on the right)

Branches (route 1)	A priori cost	Real cost (measured)	New cost after fusion (fusion α = 0.8)
a	7.4	14.8	13.3
b	5.4	10.8	9.7
c	9.8	19.7	17.7
d	8.3	16.7	15.1
e	1.0	2.0	1.8
f	4.9	9.8	8.9
g	5.9	11.8	10.6
h	5.9	11.8	10.6
i	3.9	7.9	7.1
j (incomplete)	11.8	23.6	21.3
Total path cost	64.2	129.0	116.0

Table 5. Initial, measured and fused branch weights for route 1

The results under analysis come from different performances of the same global trajectory. The area of interest, along with the start and destination points (circles on the left and the right, respectively), are shown in Figure 12. To reach the goal from the initial point, two routes can be used: route 1, through corridor A, and route 2, through corridors B and C. The initial weights of the branches are computed from their lengths as the estimated time to travel them. These cost values are computed assuming a constant speed along the whole trajectory.

Thus, the initial time cost for route 1 is 64.2 seconds and 97.0 seconds for route 2. The experiments run without people in the environment led to 68.7 seconds to complete route 1 and 107.1 seconds to complete route 2.
Under normal conditions, corridor A is full of people (i. e. moving obstacles), causing a significant increase in the time it takes for the robot to traverse it. Thus, the average time needed to complete route 1 is 129.0 seconds, while for route 2 it takes 116.0 seconds, which is much closer to the a priori forecasted time.
The fusion of the previous and the new weights is shown in Tables 5 and 6. It can be seen that the final costs of the crowded branches increase. The consequence is that further planning tasks will choose route 2 over route 1 (116.3 seconds for route 1 vs. 112.6 seconds for route 2).

Branches (route 2)	a-priori cost (in secs.)	real cost (measured)	New cost after fusion (fusion α = 0.8)
a	7.4	8.8	8.5
b	5.4	6.4	6.2
k	26.0	31.1	30.0
l	49.0	58.6	56.7
j (incomplete)	9.3	11.1	10.8
Total path cost	97.0	116.0	112.2

Table 6. Initial, measured and fused branch weights for route 2

In order to limit the influence of rare situations, a conservative fusion strategy is used in practice (see Figure 13). The progression of the fusion result for route 2 is shown in Table 7 using two different fusion ratios: conservative (20% - 80%) and sharp (75%-25%).

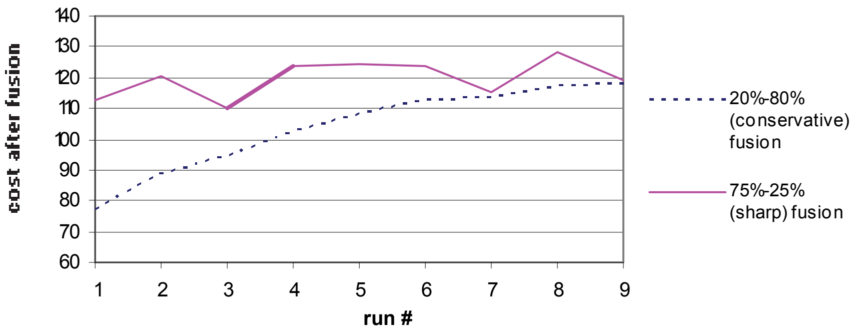


Figure 13. Graphical comparison between conservative and sharp cost fusion for route 2

Route execution	#1	#2	#3	#4	#5	#6	#7	#8	#9
Execution time (sec.)	129.0	135.2	117.4	133.9	132.1	128.7	116.1	133.2	119.5
Cost after fusion (α=20)	77.2	88.8	94.5	102.4	108.3	112.4	113.1	117.1	117.6
Cost after fusion (α=75)	112.8	120.7	110.2	124.0	124.7	123.6	115.2	128.2	118.9

Table 7. Numerical comparison between conservative and sharp cost fusion for route 2

7. Conclusions and future work

In this chapter we described a method to improve the best trajectory search for mobile robot path planning in structured environments with non-modelled obstacles. Our approach uses the experience of performed trajectories without a need to transpose information on unknown obstacles to the map for global path planning.

In the proposed algorithm, the weighting factors of the skeleton branches involved in the path are modified as a function of the traveling time. These weights are good estimators of the branch costs when the robot is moving in the presence of unknown obstacles.

Our method is especially attractive in terms of incorporating the effects of the presence of moving obstacles in the environment (for example, corridors crowded with people) in the planning process, since this obstacle information cannot be included in a map. Our approach does not eliminate the need for a local planner and its local map in our mobile robot architecture, but it reduces the number of times they will be used.

8. Acknowledgments

This work has been partially supported by the Spanish Ministry of Science and Technology, (Project DPI 2005-06210) and by the Galician Autonomous Government Department of Innovation and Industry (Program INCITE, Projects PGIDIT 07PXIB303199PR and PGIDIT06PXIC303194PN).

9. References

Diéguez, A.R. ; Raimúndez, C ; Sanz, R. ; Fernández, J.L. & Delgado, E. (1995). An intelligent supervisory model for path planning and guidance of mobile robots in non-structured environments. *Preprints of the 2nd IFAC Conference on Intelligent Autonomous Vehicles (IAV-95)*, pp. 81-86, Espoo-Helsinki (Finland), June 1995.

Diéguez, A.R. ; Sanz R. & Fernández, J.L. (2003). Deliverative On-Line Local Path Planning for Autonomous Mobile Robots. *Journal of Intelligent and Robotic Systems*, Vol. 37, 1, May 2003, pp. 1-19, ISSN: 0921-0296.

Diéguez, A.R. ; Sanz, R. & Fernández, J.L. (1998). Improving global motion planning for mobile robots by experimental measurement of traveling time. *Preprints of the 3rd IFAC Conference on Intelligent Autonomous Vehicles (IAV-98)*, 1998. pp. 403-408, Madrid (Spain), March 1998.

Diéguez, A.R. ; Sanz, R. & Fernández, J.L. (2007). A Global Motion Planner that Learns from Experience for Autonomous Mobile Robots. *Robotics & Computer Integrated Manufacturing*, Vol. 23-5, October 2007, pp. 544-552, ISSN: 0736-5845.

Fernández, J.L.; Sanz, R.; Paz, E. & Alonso, C. (2008). Using hierarchical binary Petri nets to build robust mobile robot applications: RoboGraph, Presented to *2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, May 2008.

Heero, K. (2006). *Path Planning and Learning Strategies for Mobile Robots in Dynamics Partially Known Environment*. Ph.D. Thesis, Tartu University Press, ISBN: 9949-11-308-3, Tartu, Estonia.

Hsu, D.; Kindel, R.; Latombe, J.C. & Rock. S. (2002). Randomized Kinodynamic Motion Planning with Moving Obstacles. *The International Journal of Robotics Research*, Vol. 21, No. 3, (March 2003), pp. 233-255, ISSN: 0278-3649.

Hwang, J.K. & Ahuja, N. (1992). Gross Motion Planning- A Survey. *ACM Computing Surveys*, Vol. 24, No. 3, pp. 219-291, ISSN 0360-0300.

Jaillet, J. & Simeon, T. (2004). A PRM-based motion planner for dynamically changing environments, in *Proceedings of the 2004 IEEE International Conference on Intelligent Robots and Systems* (IROS 2004), Vol. 2, pp. 1606-1611, ISBN: 0-7803-8463-6, 28 Sept.-2 Oct. 2004.

Latombe, J. C. (1991). *Robot Motion Planning*, Kluwer Academic Pub., ISBN, Norwell, MA.

LaValle, S.M. (2006), *Planning Algorithms*, Cambridge University Press, ISBN-13: 9780521862059.

Philippsen, R.; Jensen, B.; Siegwart, R. (2007). Towards Real-Time Sensor-Based Path Planning in Highly Dynamic Environments, In: *Autonomous Navigation in Dynamic Environments, Springer Tracts in Advanced Robotics*, Vol. 35, Laugier, C.; Chatila, R., (Eds.), pp. 135-148, Springer, ISBN 978-3-540-73421-5, Berlin.

Russell, S. & Norvig, P. (1995). *Artificial Intelligence: A modern Approach*. Prentice Hall, ISBN-13: 9780131038059, Engelwood Cliffs, New Jersey.

Simmons, R. G. (1994). Structured control for autonomous robots, *IEEE Trans. on Robotics and Automation*, Vol. 10, No. 1, (February 1994), pp. 34-43, ISSN: 1042-296X.

Szczerba, R.J. & Chen, D.Z. (1995). *A Time-Sweeping Approach for Determining Shortest Paths in a Dynamic*, 2-D Environment with Multiple, Moving Goals, *Computer Science and Engineering Technical Report*, TR 1995-23., University of Notre Dame, Notre Dame, Indiana.