Hierarchical Planning in a Mobile Robot for Map Learning and Navigation

Cristina Urdiales [*], Antonio Bandera, Eduardo Pérez, Alberto Poncela, and Francisco Sandoval

Dpto. Tecnología Electrónica, ETSI Telecomunicación,
Universidad de Málaga, 29071 Málaga-Spain

Abstract. This chapter focuses on autonomous navigation for mobile robots. We propose a hybrid layered architecture, which is used to navigate in totally or partially explored environments using sonar sensors. Our architecture relies on a hierarchical representation of the environment, which has both a metric and a topological level, which is based on the metric level. High level planning layers work at the topological level deliberatively, while low level navigation layers operate at the metric level reactively. The main advantage of the proposed scheme is that it can operate in both known and unknown environments rapidly and efficiently.

1 Introduction

Although robots were originally conceived as reprogrammable, multifunctional devices, mainly designed to move material in industrial environments, the envisioned potential of robotics systems has risen tremendously over the last few decades. Mobility, particularly, has been greatly improved, adding versatility to these systems. Nowadays, robots can be used for hazardous waste cleanup, spatial exploration or demining, just to mention a few applications. However, mobile robots are no longer bound to a well-known operation frame. Furthermore, they can even be meant to operate in unknown or changing environments. In these cases, robots can no longer be preprogrammed to pursue a fixed course of action like traditional factory robots. Instead, they must plan their actions according to the environment and react to potential unexpected situations [1]. To achieve this adaptive behaviour, robots usually rely on one or several types of sensors to perceive and act according to the outer world. Typically, on-board sensors for a given robot are chosen according to several criteria: field of view, range capability, accuracy and resolution, real-time operation, redundancy, simplicity, size and power comsumption. Popular sensors typically include sonar, tactile, infrared and laser sensors and videocameras. The final selection basically depends on what kind of behaviour is expected of the robot. Navigational behaviours, particularly, often rely on sonar sensors despite their obvious drawbacks. Even though sonar sensors have a wide arc of uncertainty and, in their simplest

[*] Corresponding author e-mail: cris@dte.uma.es

version, only provide information about the distance to the closest obstacle in the beam direction, they are light, cheap, fast, easy to process and have a long detection range. The advantages of using sonar sensors for navigation are widely discussed in [2].

Basically, the navigation problem consists of answering three simple questions: i) where am I? ii) where am I going? and iii) how do I get there? The answer to the first question is known as the localization problem. It involves determining the agent's position according to what it perceives and where it was previously believed to be. This problem is typically solved by measurement, correlation and triangulation. The second and third questions involve determining a goal and planning a path that leads to the goal. They basically concern path planning and collision avoidance. Hence, the navigation problem can be subdivided into three tasks: i) collision avoidance; ii) robot positioning; and iii) path planning.

The localization problem is not easy to solve. Basically, most systems rely, at least partly, on odometry. However, robot slippage provokes small positioning errors that accumulate unrestrainedly. Consequently, after a while the robot may not know its real position. The problem is even worse if no odometric information is available. In this case, the problem is known as global localization, while localization based on odometric information is known as tracking. Most tracking approaches rely on Kalman filtering to integrate sensor information over time. Global techniques aim instead at locating significant features (landmarks) in the environment in order to determine the robot position with respect to those landmarks. Even though localization is of capital importance to navigation, this chapter does not cover the issue, because, in our case, we rely on well-known localization techniques plus a compass to correct odometric information. Further information on localization can be found in [2] and [3]. Instead, this chapter focuses on techniques to determine a goal and to plan how to get there. Our system is meant to operate in real time even in unknown or changing environments.

The chapter is organised as follows. First, Section 2 presents an overview of classic navigation control techniques and a proposal for a hybrid control architecture meant to combine efficient planning and fast reactive behaviours. Section 3 proposes a new model of the environment, which is required to implement the proposed architecture. The main novelty of the proposed model is that it efficiently combines a topological and a metric representation to allow hierarchical planning. Then, Section 4 presents a navigation technique based on the proposed architecture. This technique allows the robot to move flexiblely and efficiently in order to visit one or more places in the environment. Section 5 presents several results for a real robot running on the proposed scheme. Finally, conclusions and future work are presented in Section 6.

2 A Hybrid Approach to the Navigation Problem

Navigation is clearly a central concern for mobile robotics and, consequently, many navigation techniques have been proposed. Nevertheless, navigation control schemes can be broadly divided into two large groups [4]: reactive and deliberative schemes. Initially, most navigation schemes were based on deliberative planning [5]. Deliberative planning typically relies on a classical top-down methodology known as horizontal decomposition [6][7]. In these cases, the world is represented and processed according to actions and events in a $sense-model-plan-act$ cycle. These schemes are typically decomposed hierarchically into several levels, which interchange information by means of well-defined flow paths. While the higher levels are in charge of planning and reasoning, the lower levels support low-level control and direct hardware actions. Briefly, these schemes have the following features:

- A hierarchical structure with well-defined functions for each level.
- Communication between levels is predictable and deterministic.
- The upper levels in the hierarchy decompose each task into subtasks for lower levels.
- The lower levels work locally.
- They are strongly dependent on representations of the environment.

Deliberative schemes are often criticized for their inability to react rapidly. It can be easily observed that the robot must sense, model and plan before acting in these schemes. Although there have been some attempts to fix this latency problem using temporal constraints [8], these schemes have a second problem: they traditionally assume that the environment remains almost static between consecutive observations. In this case, any condition violating this assumption, like mobile or unexpected obstacles, may cause problems for the robot.

To overcome the drawbacks of deliberative systems, reactive schemes rely on directly coupling sensors and actuators [9]. The reactive paradigm is based on animal intelligence models, and it basically produces a global action by combining one or more reactive behaviours. This action is known as emergent behaviour. Unlike classic deliberative systems, reactive schemes can easily deal with several sensors, as well as aiming for several goals. Besides, they are quite robust against sensor errors and noise, and they can be easily modified to deal with changes in hardware or tasks. Briefly, reactive schemes are preferable to deliberative schemes when [4]:

- The environment is dynamic.
- Inmediate robot sensing is adequate for the task at hand.
- The robot is not easy to locate with respect to a global coordinate system.
- No valid representation of the environment is available.

The best known reactive scheme is the subsumption architecture [10], which consists of a number of behaviour modules arranged in a hierarchy.

Different layers of the architecture are responsible for different behaviours, which are known as levels of competence. A level of competence may be, for example, to avoid contact with objects, to wander through the environment without hitting things or to build a map. Each level of competence includes all the earlier ones. The final emergent behaviour of the system is the set of reactions that emerge from the modules designed to achieve the different levels of competence. Unfortunately, reactive schemes also have important drawbacks. First, emergent behaviours may be very unpredictable. For this very reason, scheme performance may become inefficient in some cases. They are also typically prone to fall into local traps. Finally, they are quite difficult to debug.

Hybrid systems combine deliberative and reactive schemes in order to achieve better performance. Usually, low level control is performed reactively, whereas high level processing follows a deliberative pattern. Not only are hybrid systems supported by biological evidence [4], but they are also capable of providing efficient navigation in dynamic and totally or partially unknown environments. The main concern when building a hybrid scheme is to find the right boundary for the subdivision of functionality. Lyons [11] proposes three different ways of integrating planning and reaction:

- Hierarchical integration, where planning or reacting depend on the situation at hand.
- Planning to guide reaction, where planning modules are used to configure and set parameters for the reactive control system.
- Coupled planning-reacting, where planning and reacting are concurrent activities, each guiding the other.

Basically, according to the way in which integration is implemented, the most representative hybrid architectures are [4]: i) selection, where deliberative modules configure which reactive modules should be active; ii) advice, where deliberative modules just offer advice that reactive modules may or not may accept; iii) adaptation, where deliberative modules modulate reactive modules according to the world and task requirements; and iv) postponement, where deliberative modules work only when necessary.

Most recent approaches to navigation rely on hybrid architectures because of their advantages with regard to purely reactive or deliberative ones. Thus, in this Section we propose a hybrid architecture to control a sonar-based mobile robot. Even though there are several guidelines for building an efficient architecture for robotic control, most schemes tend to be built *ad hoc*, because a given architecture is heavily influenced by hardware conditions, goals and algorithmic criteria. An excellent review of the most general approaches for build a robotic architecture can be found in [12].

One of the most pressing issues when designing a control architecture is how to manage the growing complexity of interactions both between the system and its environment and among the individual components of the

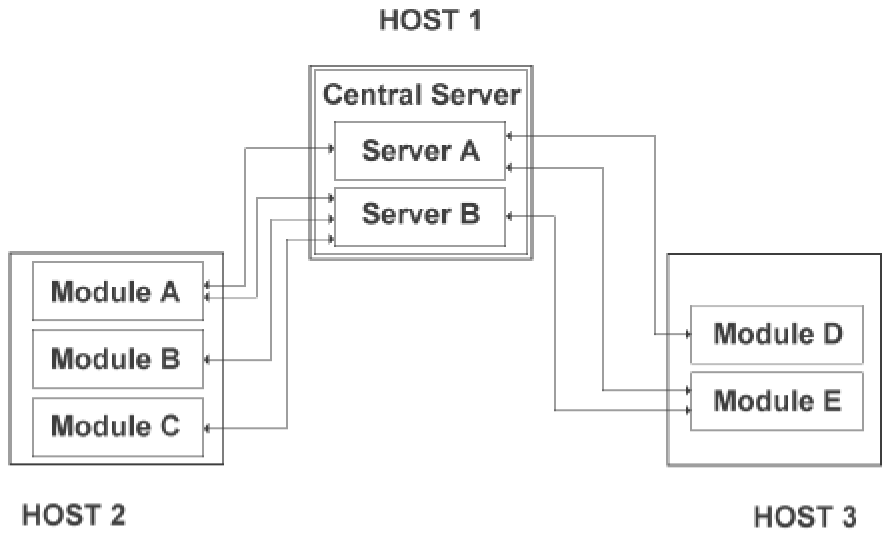

Fig. 1. Proposed architectural style.

system. Modern robotics systems, usually requiring concurrent embedded real-time performance, cannot be operated using conventional programming techniques. The computational concepts underlying a given system are referred to as architectural style [12]. One of the most popular styles for dealing with complexity is to decompose the system into modules which operate concurrently. Each of these modules has a different function, and they exchange information to work as a whole. When a system is decomposed into modules, it is necessary to provide the mechanisms required by the different modules to exchange information in order to cooperate in parallel. There are several approaches to this problem, like end-to-end connections between modules [13], specific routing agents [14] [15], or shared memory-based systems [16]. We have chosen a shared memory-based approach because it suited our scheme better than the others. In our case, the different modules of the architecture are distributed over different machines, which may even have different operating systems. Thus, we used a scheme proposed by Dulimarta [16], which was very suited for transparent information exchange between different machines and operating systems. In this scheme, modules share information by sending it to a specific data server. This server is in charge of controlling the access to all shared data. Thus, when a module needs information, all it has to do is place a request with the server. However, Dulimarta proposed a single server to manage all the information flow. Thus, when many modules send or request information simultaneously, they are queued and response

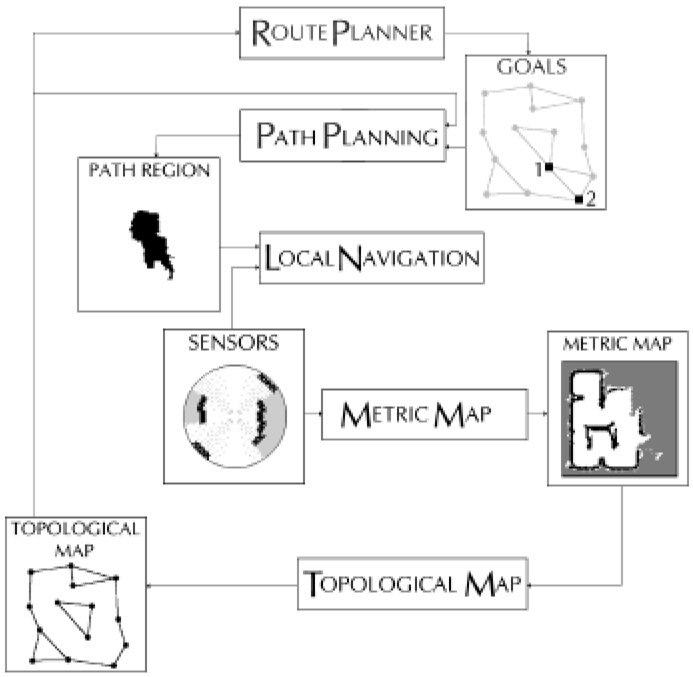

Fig. 2. Proposed architecture

time is increased. Hence, we have modified the original scheme by adding local data servers which are controlled by a central server. When a module requests information from the central server, it redirects the request to one of the local servers. Hence, it is free to receive more requests.

Now we have the architectural style, the next thing is to design the architectural structure. Basically, any given robot needs to accomplish a set of tasks, which must be decomposed into modules. Different applications may need to be decomposed differently and, since systems tend to grow, a structure needs to be flexible enough to accommodate different decomposition strategies. Often, system decomposition is hierarchical, because this approach leads to more modular systems. Indeed, quite a popular structure for navigation is the navigation hierarchy [17], because: i) each level of the hierarchy can be tested experimentally; and ii) it is valid for representing biological and robotic navigation. In this hierarchy, navigation behaviours are classified according to the complexity of the task which they can perform. While local navigation behaviours require only recognition of the goal location, way-finding behaviours require knowledge about several places, as well as their interrelation. We propose a structure based on the navigation hierarchy, where different layers are in charge of progressively less complex behaviors (Fig. 2).

Basically, high level layers are in charge of reasoning about the environment and planning a course of action. Different planning algorithms are used for this purpose, depending on the goal to be achieved. Planning algorithms require models of the environment about which they reason. Thus, on-board sensor readings are used to build these models. The processes required to model the environment are discussed in detail in Section 3. In our case, two different models are required: geometric and topological maps. The advantages of this approach, along with a brief review of other models, are presented in the next Section. We will see that these models are built by means of two different layers (Fig. 2). One layer is in charge of creating and continously updating a geometric map using the sonar readings. A second layer is in charge of creating a topological map. This second layer uses the geometric map as an input. Thus, we keep the topological map grounded.

The highest level layer of the whole structure is the route planner. This layer is in charge of determining which areas of the environment are going to be visited and in which order. If our goal is just to move from one place to another, the output of this layer is a single goal, which is equal to the final location. If, instead, the goal is to visit a set of places, the output is an ordered list of goals, which is created by minimizing factors like the total distance traveled. Then, the path planning layer uses the topological map as an input to calculate an obstacle-free path between any two locations. Finally, the local navigation layer relies on reactive navigation to track the paths calculated by the path planning layer. Section 4 focuses on the practical implementation of all these layers. Note that all these layers share information using the above architectural style. Each time a layer requires information, it sends a request to the central server. This server routes the request to the respective local server, which, in turn, sends the most recently updated data to the querying layer.

The proposed hybrid architecture works efficiently for several reasons: i) deliberative layers propose efficient paths to a goal; ii) these paths are quickly and reactively tracked to handle unexpected situations; iii) the environment is represented at two hierarchical levels so that it can be both updated and processed rapidly; iv) the local navigation layer can work even when the output from higher level layers is outdated. Thus, the robot does not need to stop when a given path needs to be recalculated due to unexpected obstacles in the way.

3 Map Generation

Representations of the environment are usually built according to either the metric or the topological paradigm. Metric approaches generate representations that explicitly reproduce the metric structure of the environment. Metric maps can be either geometric or grid-based. In geometric representations, world features, like walls or corridors, are directly mapped with respect to a global coordinate system. In grid-based representations [18], each grid cell is associated with a specific position in the environment, which is implicitly given by the cell coordinates (x, y). Any cell yields an occupancy value that is equal to the probability of that cell being occupied according to current and past sensor readings. Topological approaches [19] aim at representing the environment as a set of meaningful regions. These regions are usually represented as nodes, which are inserted each time the robot sensors perceive a pattern corresponding to a representative place. Thus, the map becomes a topological graph and nodes become linked if the robot can find an obstacle-free path between them.

Both approaches have complementary strengths and weaknesses [20]. Of all the metric maps, evidence grids have become popular because they are usually fast and easy to build and update and they provide efficient space-time integration of sonar sensors. Besides, the geometry of the grid directly corresponds to that of the real environment, so the robot position within the model can be determined by its position and orientation in the real world. However, most metric approaches rely heavily on dead-reckoning, which is unreliable in large environments. In addition, a grid is only reliable if it is highly decomposed. Hence, these approaches usually involve a huge data load and, consequently, they are computationally expensive to process when medium or large environments are modeled. On the other hand, topological maps are usually more compact, because their resolution is determined by the complexity of the environment [20]. Topological maps are primarily used for robot position estimation and path planning [21], because they allow fast planning and provide more natural interfaces for human instructions. However, topological representations also have several problems. The most important one is known as the dissambiguation problem, which involves distinguishing different places that are characterised by the same sensory pattern. Similarly, it is necessary to determine whether or not two nodes associated with an equal sensory pattern actually correspond to the same place. Odometric information is usually added to topological maps to overcome these problems. However, odometry is not always reliable and maps may become erroneous when updated or after the robot has been moving for a while. Besides, the dissambiguation process tends to be computationally expensive.

Recently, there have been several proposals for combining metric and topological representations so that their advantages can be combined as well. Basically, there are two ways of constructing a topological-metric representation: either a topological representation is annotated with metric information while it is constructed [22] or a topological map is extracted from a metric one [23] [20] [24]. The main problem of approaches relying on annotating topological maps with metric information is that acquired maps usually require further processing. This processing may be intensive and must often be performed off-line. Extracting topological from metric maps appears to be easier. This type of schemes typically rely on splitting metric maps into a set of homogeneous regions, which are the nodes of the graph.

We propose a new mixed approach for integrating the metric and topological paradigms [25]. Our method is related to proposals by Thrun [20], Arleo [23] or Zelinsky [24] in several respects. First, all these methods use a local occupancy grid to model the region surrounding the robot at the metric level. Some of the above-mentioned methods use local grids to model obstacle boundaries by means of straight lines [23]. Others, like ours, just use these local grids to construct a global one [20]. Despite these similarities, all these approaches use quite different methods to construct the topological map. Thrun's method [20] extracts the map off-line using Voronoi diagrams and only after the whole global map is available. Arleo [23] splits the metric map into rectangular regions, and this method cannot deal with irregular shaped regions. Zelinsky [24] performs this partitioning using quadtrees. However, the optimality of the resulting partition is strongly dependent on the obstacle distribution of the environment. Our approach extracts the topological map from the metric representation on-line. Besides, it can deal with irregular regions and it does not have the disadvantages of quadtree decomposition. The following two subsections describe the metric map learning process and the topological map extraction processes, respectively.

3.1 Metric Map Learning

In this chapter, the metric map is a two-dimensional occupancy grid, similar to the one originally proposed by Moravec and Elfes [18]. Each grid cell (x, y) in the map yields the occupancy probability of the respective region of the environment. These cells are modified according to the readings of Polaroid sonar sensors, which have an arc of uncertainty of approximately 25^o. A very simple probability distribution is used to model the cells in a sonar scan. The occupancy probability of a cell is modelled as:

$$P(\theta, \rho) \begin{cases} -\epsilon, & \text{if } 0 \leq \rho < d - \delta, 0 \leq |\theta| \leq \beta/2 \\ +\varphi, & \text{if } d - \delta \leq \rho < d + \delta, 0 \leq |\theta| \leq \beta/2 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

ρ and θ being the distance to the robot and the angle of the main axis of the sonar beam, respectively; d the range measurement returned by the sonar sensor and β the beam aperture. $2 \cdot \delta$ determines the width of the region of uncertainty where the obstacle could be located. Finally, the empty and occupied probability density functions for a cell inside the sonar beam are given by constants ϵ and φ, respectively. The resulting probability distribution for a single scan is shown in Fig. 3.

To build the metric map, sonar readings are integrated into a local occupancy grid, which is the result of the weighted addition of all the sonar sensor models. Fig. 4a shows an example of a sonar scan and its interpretation. The

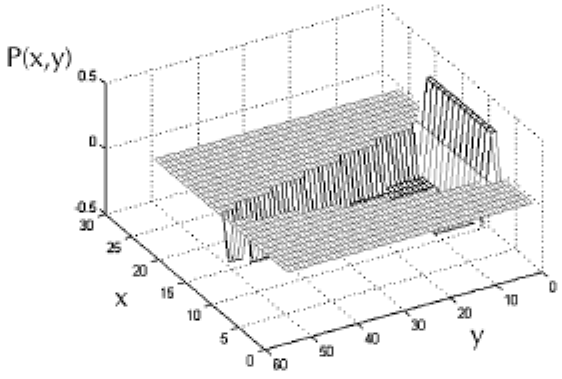

Fig. 3. Current model of the sonar sensor.

darker the value in the circular region around the robot, the larger the internal computed value. Note that the local grid has negative and positive values, which correspond to empty and occupied regions, respectively.

Sonar interpretations must be integrated over time to update the map coherently. For this purpose, the local grid adquired at time t (P^t_{local}) is combined with the available metric map (P^{t-1}_{metric}). In our case, the metric map cells coinciding with the local grid region are updated using the expression:

$$P^t_{metric} = P^{t-1}_{metric} + P^t_{local} \tag{2}$$

The main advantage of the above sensor model and integration algorithm is their low computational complexity. As a field test, the proposed method, the Dempster-Shafer approach [26] and Bayes' method [20] have been used to build different maps of a large real environment, consisting of a corridor and a room of irregular shape. Fig. 4b represents a map created using the proposed approach. The CPU load in the proposed method is 15 % lower than in the other two methods.

Finally, it is important to note that the precision of the resulting metric map depends on the correct alignment of the robot with its map. Hence, slippage and drifting must be detected and corrected [27]. This information is extracted from the localization layer, which uses well-known techniques like correlation of the local map and the respective section of the global map [28].

3.2 Topological Map Building

A new hierarchical structure is proposed in order to extract a topological map from the above metric map. This structure is constructed as follows:

1. Metric map thresholding. Initially, the occupancy value of each cell in the metric map is thresholded. Cells whose occupancy value is below threshold U_1 are considered free space ($P(x,y) = c_F$). Cells whose occupancy

value is above U_1 and below threshold U_2 are considered unexplored ($P(x,y) = c_N$). All other cells are considered occupied ($P(x,y) = c_O$).

2. Hierarchical structure generation. The thresholded metric map becomes the base of a pyramidal structure. Each level l of this pyramid is a reduced map with 1/4 of the cells of the level immediately below. Each pyramid cell (x,y,l) has five associated parameters:
   - Homogeneity, $H(x,y,l)$. $H(x,y,l)$ is set to 1 if the four cells immediately underneath have the same occupancy probability and their homogeneity values are equal to 1. Otherwise, it is set to 0.
   - Occupancy probability, $P(x,y,l)$. If the cell is homogeneous, $P(x,y,l)$ is equal to the occupancy probability value of any of the four cells immediately underneath. If the cell is not homogeneous, the value of $P(x,y,l)$ is set to a fixed value (c_{NH}).
   - Area, $A(x,y,l)$. It is equal to the addition of the areas of the four cells immediately underneath.
   - Parent link, $(X,Y)_{(x,y,l)}$. If $H(x,y,l)$ is equal to 1, the values of parent link of the four cells immediately underneath are set to (x,y). Otherwise, these four parent links are set to a null value.
   - Centroid, $C(x,y,l)$. It is the centre of mass of the base region associated with (x,y,l).

   After generation, the cells at upper levels of the hierarchical structure have an homogeneity value equal to 1. These cells can divide the grid-based map like quadtree approach does [29]. However, the complexity of this decomposition is not directly related to the world complexity. Instead, it depends on the position of the obstacles.
3. Homogeneous cell fusion. In this step, the algorithm tries to link cells whose parent link values are null. Basically, these cells, (x,y,l), are linked to parents of neighbouring cells, $(x_p,y_p,l+1)$, if the following conditions are true:

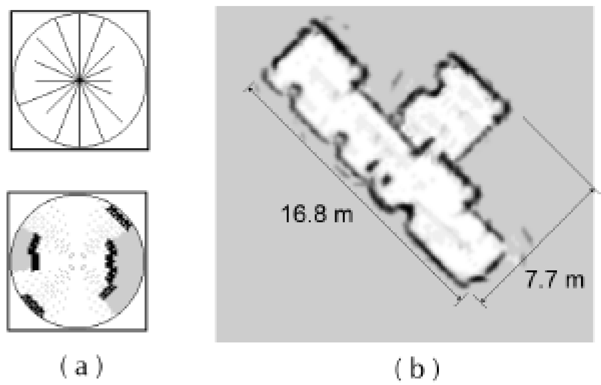

Fig. 4. a) Sensor interpretation: sonar scan (top) and local map (bottom); and b) grid-based map created using the proposed approach.

- $H(x,y,l) = 1$ & $H(x_p, y_p, l+1) = 1$
- $P(x,y,l) = P(x_p, y_p, l+1)$
- $||C(x,y,l) - C(x_p, y_p, l+1)||_2 < DistMax$,
  $DistMax$ being a threshold that establishes the maximum dispersion of the regions at the base.

4. Homogeneous cell classification. Two neighbouring cells, (x_1, y_1, l) and (x_2, y_2, l), are fused if the following conditions are true:
   - $(X,Y)_{(x_1,y_1,l)} = NULL$
   - $(X,Y)_{(x_2,y_2,l)} = NULL$
   - $H(x_1, y_1, l) = 1$ & $H(x_2, y_2, l) = 1$
   - $P(x_1, y_1, l) = P(x_2, y_2, l)$
   - $||C(x_1, y_1, l) - C(x_2, y_2, l)||_2 < DistMax$

The proposed algorithm extracts the topological map from the metric representation on-line. Besides, unlike other approaches, the whole global map does not have to be acquired [20]. Fig. 5 illustrates the process of extracting a topological map from a grid-based map. Fig. 5a shows the thresholded map associated with the map in Fig. 4b. The resulting partitioning is shown in Fig. 5d. Note that cells yielding an area less than 8 are not shown. Finally, Fig. 6 shows the final topological graph. The main advantages of the proposed topological graph extracting algorithm are its low computational load and that it depends only on threshold $DistMax$. The average topological graph extracting times are shown in Table 1.

In these tests, metric maps have 65,536 cells. The low computational time allows the on-line generation of the proposed topological map.

Fig. 7 illustrates the algorithm dependence on threshold $DistMax$. If $DistMax$ is large, the regions tend to be large, but if it is low, the resulting map is excessively partitioned. Fortunately, it is very easy to choose a suitable $DistMax$ and it has been empirically proven that $DistMax \in [50, 70]$ works correctly in most cases.

4 Survey Navigation

Survey navigation is the highest level of the navigation hierarchy [17]. In fact, there are many examples of lower level navigation mechanisms, like local navi-

Table 1. Topological map extracting times

Process	Time (typical)
Structure initialisation	0.10 s
Homogeneous cells linking	0.05 s
Node definition	0.05 s
Total time	0.20 s

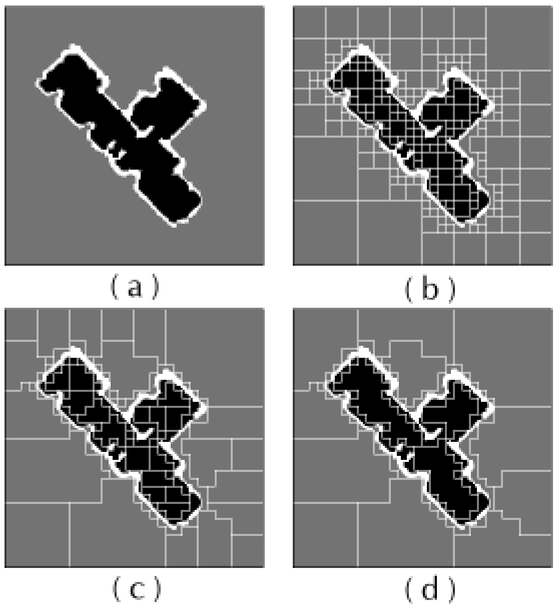

Fig. 5. Extracting the topological map: a) thresholded map; b) base regions generated after the generation step; c) base regions generated after homogeneous cell fusion; and d) base regions generated after homogeneous cell classification (topological regions).

gation behaviours, recognition triggered responses and topological navigation in the animal kingdom, but survey navigation may be limited to vertebrates. Survey navigation specifically requires embedding all known places and their spatial relations into a common framework of reference so that the representation can be manipulated as a whole. When every location is embedded into a common framework of reference, the agent can find novel paths over unknown terrain by inferring its spatial relation to known places. There is

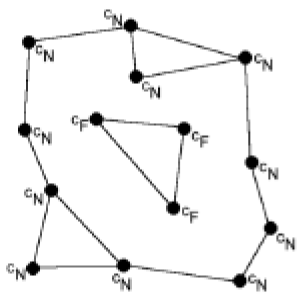

Fig. 6. Topological graph associated with the metric map in Fig. 4.b.

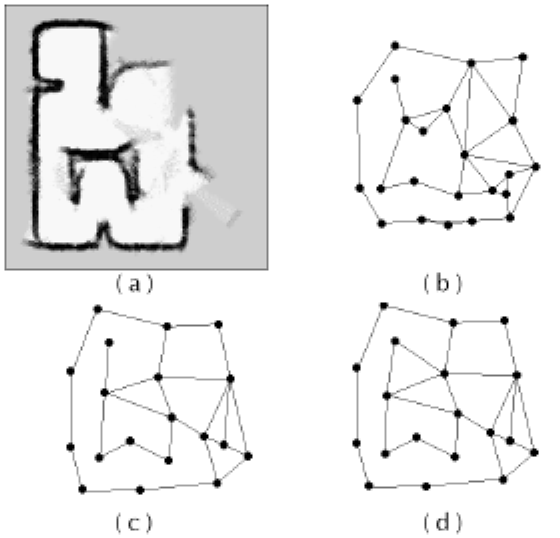

Fig. 7. Threshold $DistMax$ evaluation: a) metric map; b) topological graph ($DistMax = 40$), c) topological graph ($DistMax = 60$), and d) topological map ($DistMax = 80$).

not a lot of work on survey navigation. In fact, existing approaches to planning in partially unknown environments rely on metric maps (i.e. [18]), where geometric relations between places are explicitly represented analytically. As mentioned above, metric maps are easy to build and fast to update, but they are also sensitive to all errors affecting metric information, namely robot slippage, and they typically yield a very large data volume. Hence, planning over an average sized environment may be so costly that on-line operation is not possible. Classic topological maps do not include information on unexplored areas, so they are not suitable for this purpose either. However, a combined topological-metric structure like the one proposed can be used for survey navigation. Hence, we propose a navigation scheme for performing survey navigation on a totally or partially unknown environment.

This Section presents the practical implementation of the layers of the architectural structure in Fig. 2 to achieve survey navigation. The metric map layer is in charge of creating and updating the evidence grid. This layer receives the position information from the localization layer and the sonar readings to update the grid each time the sonars are read. The only output of this layer is a global evidence grid (Fig. 8a). It must be noted that the size of the world is not known *a priori*, so this grid may grow in terms of x and y. The grid is the input to the topological map layer, which is in charge of creating a topological map as explained in the previous Section. Even though the proposed topological map can be created rapidly, it cannot be generated as fast as grids are updated. Thus, this layer can be triggered according to two different approaches. If the topological map needs to be updated as often as possible, each time a topological map is finished, the layer starts to create a new one. Obviously, if a planning layer requests a topological map while it is still being created, it will receive the former topological map. However, topological maps are created rapidly, so they are never very outdated. The main problem with this approach is that the resulting computational load may be excessive, specially considering that the topological map does not usually change so often. A second option is to shoot the layer only when the planning module, which uses the topological map as an input, is going to be triggered. The main disadvantage of this approach is that the planning module has to wait for the topological map layer to finish before starting to plan, because, in this case, a given topological map may be quite outdated. These two options may be combined efficiently by predicting how often the environment is going to change in the near future: for example, the metric map is likely to change a lot in unexplored areas. In our current model, the system is only meant to navigate. Thus, the topological map is updated as often as possible. Fig. 8b shows a topological map created by this layer.

As discussed in Section 2, the route planner is in charge of determining which areas of the environment are going to be visited and in which order. Hence, its output is an ordered list of goals which is created by minimizing factors like the total distance traveled. The available topological map is fed to the layer in order to calculate this list. The optimization problem is handled as a classic Traveling Salesperson Problem (TSP). The TSP involves search for the shortest tour of all the nodes, given a finite set of nodes $N = c_1, c_2, ..., c_n$ and a distance $d(c_i, c_j)$ for each pair of nodes. Note that the TSP is one of the most representatives NP-complete problems. Hence, its processing time is drastically increased along with the problem instance. Because of this, the route planner works with the topological map, which yields fewer nodes, rather than with the metric map. Being a well-known problem, many methods have been proposed to solve the TSP. Although the TSP has been traditionally used as a neural network benchmark, its validity for this particular problem has been widely questioned [30]. Thus, we use the method proposed in [31] to solve it. This method is based on a genetic algorithm and its advantages and drawbacks are widely discussed in [31]. Note that this layer only provides the order in which the resulting goal locations should be visited and not a path between them.

The path planning layer is used to calculate an obstacle-free path between two consecutive goal locations of the set given by the route planner. First, a path between the current position of the robot and the goal node is calculated using the topological map as the problem instance. This calculation is performed using the A* algorithm, which is very fast when dealing with a small number of nodes. Then, the resulting node path is propagated to metric level by means of the link structure. At the metric level, the node path is a region of free cells between the robot and the goal location. Fig. 8b shows

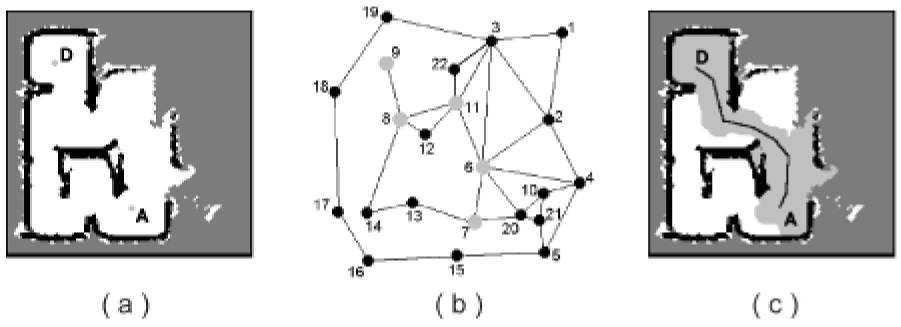

Fig. 8. Path planning: a) metric map (D=departure location; A=arrival location); b) node path at topological level; c) resulting path at metric level

a path calculated at the topological level. Nodes belonging to that path are marked in gray. Fig. 8c shows the path region at metric level, also in gray. The path planning layer is simply providing an efficient path region to the goal. We are not interested in calculating a precise path to the goal, because such a path might be unfeasible in the presence of unexpected obstacles.

We use a potential field approach to implement a low level navigation layer to move along the path provided by the path planning layer. Potential fields were originally proposed by Latombe [32] and they consist of modelling obstacles as repulsors and goals as attractors. The robot moves according to the vector resulting from its position in the field. Note that this approach has several problems, as reported in [33]. Recent proposals to solve these problems [34] [35] rely on more deliberative approaches. Our system relies on modulating low level navigation using the results of the deliberative path planning layer. In this case, the boundaries of the path region, as well as the boundaries of any obstacle inside the path region, become repulsors, while the goal is an attractor. Fig. 8c shows the final path output by the local navigator for the region corresponding to the node path in Fig. 8b. It is important to note that the map of the environment is updated while the robot is navigating. Thus, unexpected objects appearing in the path region may make it impossible to reach the goal. In such cases, the robot keeps moving reactively until a new path region is returned by the path planning layer. It should be noted that, in some cases, the unexpected obstacles could block the path to the goal. In these cases, the route planner can provide a different goal for the path planning layer. Thus, a path region from the current position of the robot to the new goal would be calculated.

The first layer to be triggered when the robot starts to navigate is the route planner. This planner provides a set of goal locations according to the available information about the environment. It should be noted that part of the environment may be unexplored or changed, so this information is likely to change when the robot starts to move. When a set of goals is

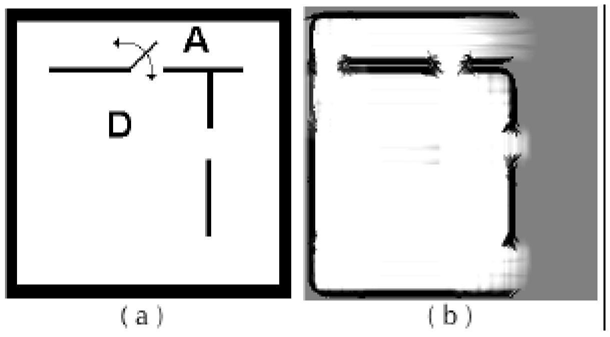

Fig. 9. A survey navigation problem: a) emulated environment with a closing door; b) map available before the closed door is detected.

available, the path planning layer calculates a trajectory between the first two goals. If there is no trajectory to move from one to the other, the route planner must be triggered again. Otherwise, the local navigator tracks the resulting route until it arrives at the goal location. While the robot moves, the environment is being explored and maps are being updated. Thus, if the local navigator cannot reach the goal location, a new path is calculated between the current position of the robot and the goal. It should be noted that the path planning layer is very fast as long as a valid topological map is available, so the new trajectory can be calculated on-line. If no valid trajectory can be found between both locations, the route planner is triggered.

Fig. 9 presents a typical problem for the proposed scheme working on a simulator. The 1056 m^2 environment in Fig. 9a is only partially explored by the robot. Fig. 9b presents an occupancy grid yielding 256x256 cells where obstacles are black, free space is white, and unexplored areas are gray. Initially, the door is open. Thus, the easiest way to go from point D (departure) to point A (arrival) is through the door. Then, we close the door. A recognition triggered response based scheme would fail, because the response to the input sensory pattern at point D would always be to go through the door. A topological approach would find a route across the west side of the room, because no nodes or arcs are available for the unexplored area. Path planning at the metric level would deal with a problem instance of 65536 cells.

Fig. 10 presents the results of applying the proposed survey navigation scheme for the problem in Fig. 9. Three different situations have been tested to prove the efficiency of the scheme. Fig. 10a shows the initial topological metric map when the door is still open. Topological nodes are marked with circles. It should be noted that if there is an arc between two nodes, it means that there is a feasible path from one node to the other. However, the path is not a straight line. It can also be observed that unexplored areas are also represented at the topological level, because their geometric relations to the other nodes can be extracted from the associated metric map. Hence, even

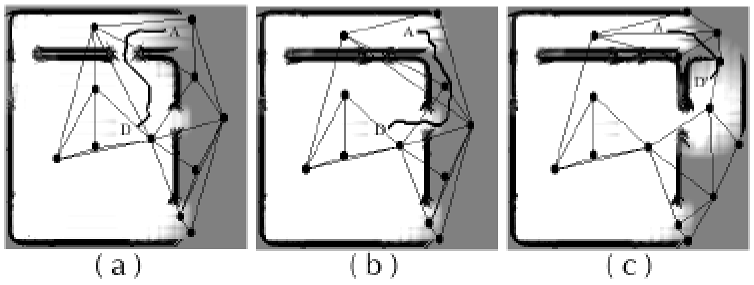

Fig. 10. Maps and resulting paths for: a) original situation with open door; b) situation a) with closed door; c) unexpected obstacle in unexplored regions.

though the robot has not traveled across the east side of the map, it is feasible that there might be a path from D to A crossing the unexplored area. Nevertheless, as long as the door is open, the shortest path from D to A runs through the door as expected. In Fig. 10b, the door is closed. Hence, a new path is required to reach the goal. Note that the metric map is updated to include the closed door. Consequently, the associated topological map is also modified and the path in Fig. 10a is no longer feasible. Note how the upper part of the topological graph is modified to represent this situation. Fig. 10b shows the new route chosen by the robot to reach A. Note that the route crosses the unexplored area, because it would be more costly to border the north wall. Finally, Fig. 10c illustrates a typical problem when planned routes go through unknown regions: there is an unexpected obstacle in the unexplored area that makes the path in Fig. 10b unfeasible. Since the cognitive map can be updated very quickly and path planning based on the topological map is fast as well, the robot can calculate a new route and still reach its goal more efficiently than if it had chosen to border the north wall.

5 Experiments and Results

The proposed survey navigation scheme has been implemented on a Pioneer 2-AT mobile robot in order to test its validity in real conditions. This Pioneer robot is a four-wheeled robot having 6 front sonar sensors plus 2-side sonar sensors. It has an on-board 400 MHz Versak6 PC with PC104+ bus and 32 Mb RAM. This robot is connected to an on-board laptop which is linked to a local area network by means of a 11 Mbps Ethernet link. The architecture central server runs on a 166 MHz Pentium PC with 32 Mb RAM. The other machines in the LAN are also ordinary PCs.

It should be noted that real environments are quite noisy from the sensor point of view. Sonar readings are affected by refractions, reflections and multiple echoes and, consequently, the adquired models of the environment are not as well defined as they were in simulation.

A first test in a small environment is presented in Fig. 12. This environment is the laboratory shown in Fig. 11a. Even though it is not a large place, different kinds of materials, like wood, metal cupboards or glass, as well as tables and chairs, are present. Initially, the robot starts at the northeast corner of the room. After a few movements, the metric map of the place shown in Fig. 12a is available. This map presents 256x256 cells, and its corresponding topological map is presented in Fig. 12b. Fig. 12c shows departure and arrival points (D and A, respectively). D is equal to the current position of the robot and A is equal to the desired goal. Using the topological map, the path planning layer returns a path region to the local planner. Fig. 12c shows the path that the robot would follow according to the local navigation layer. It should be observed that this is a preliminary path, because the final path is influenced by on-going sensor readings.

In order to test the robot's ability to react to unexpected obstacles in its way, a person walked in front of the robot when it was following the path in Fig. 12c. Fig. 12d shows a metric map in which the mobile obstacle is already mapped. Since the former path is unfeasible, the path planning layer calculates a new one using the updated topological map in Fig. 12e. Fig. 12f shows the final path that the local navigation layer plans to follow.

Table 2 shows the processing times of the algorithm on a Pentium 550. It should be noted that no route planning is necessary in these cases, because only one node is going to be visited. Nevertheless, we forced the route planning layer to work to evaluate its processing time. Note that processing times are lower than a second. Even though there is an upper bound of approximately 0.25 s on topological map extraction for 256x256 grids on a PC like the aforementioned one, processing times may vary depending on the layout of the grid, and the topological map is generated much faster if the structure of the environment seems to be easier to analyze.

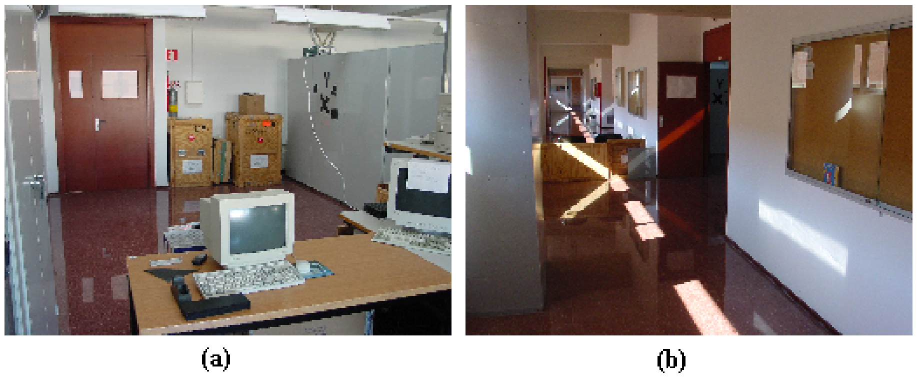

Fig. 11. Real environments.

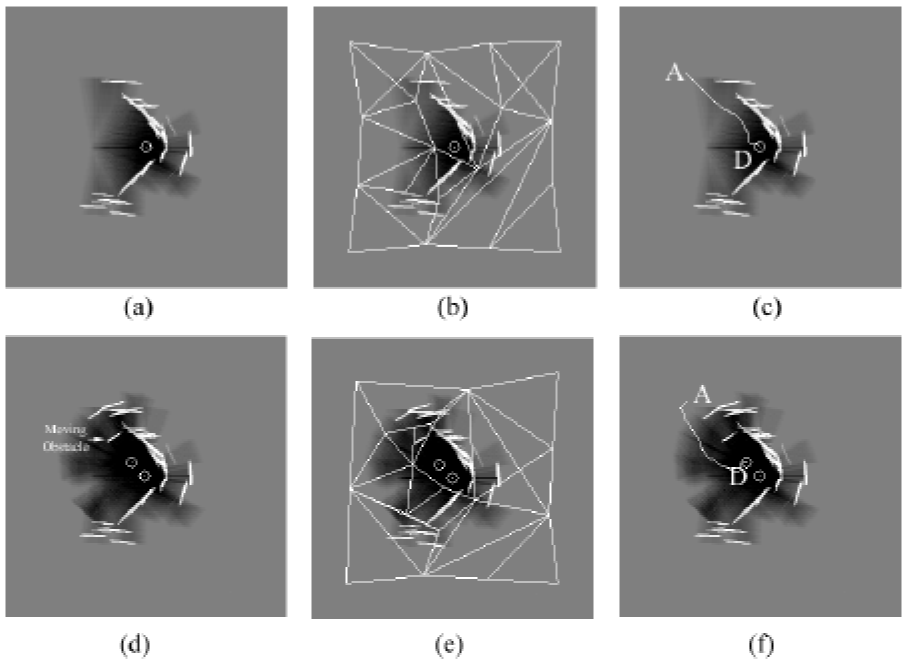

Fig. 12. Point to point navigation: a) original metric map; b) original topological map; c) original route proposed by the local navigator; d) new metric map; e) new topological map; f) final route to goal.

A second test environment is presented in Fig. 13. This environment is the corridor in Fig. 11b, a larger place than the laboratory shown in Fig. 11a. Figs. 13a,d,g,h represent the metric map with 256x256 cells, Figs. 13b,e,h,k represent the topological map, and Figs. 13c,f,i,l represent the original route proposed by the local navigator. As shown in Fig. 13a the initial metric map is completely unknown, the initial topological map of Fig. 13b has four nodes, and the robot begins to move from the actual position (departure point D) to one node of the topological map (arrival point A) in Fig. 13c. Some time

Table 2. Processing times for navigation in a small environment

Process	Time (typical)
Topological map building	0.20 s
Route planning	0.20 s
Path planning	0.10 s
Total time	0.50 s

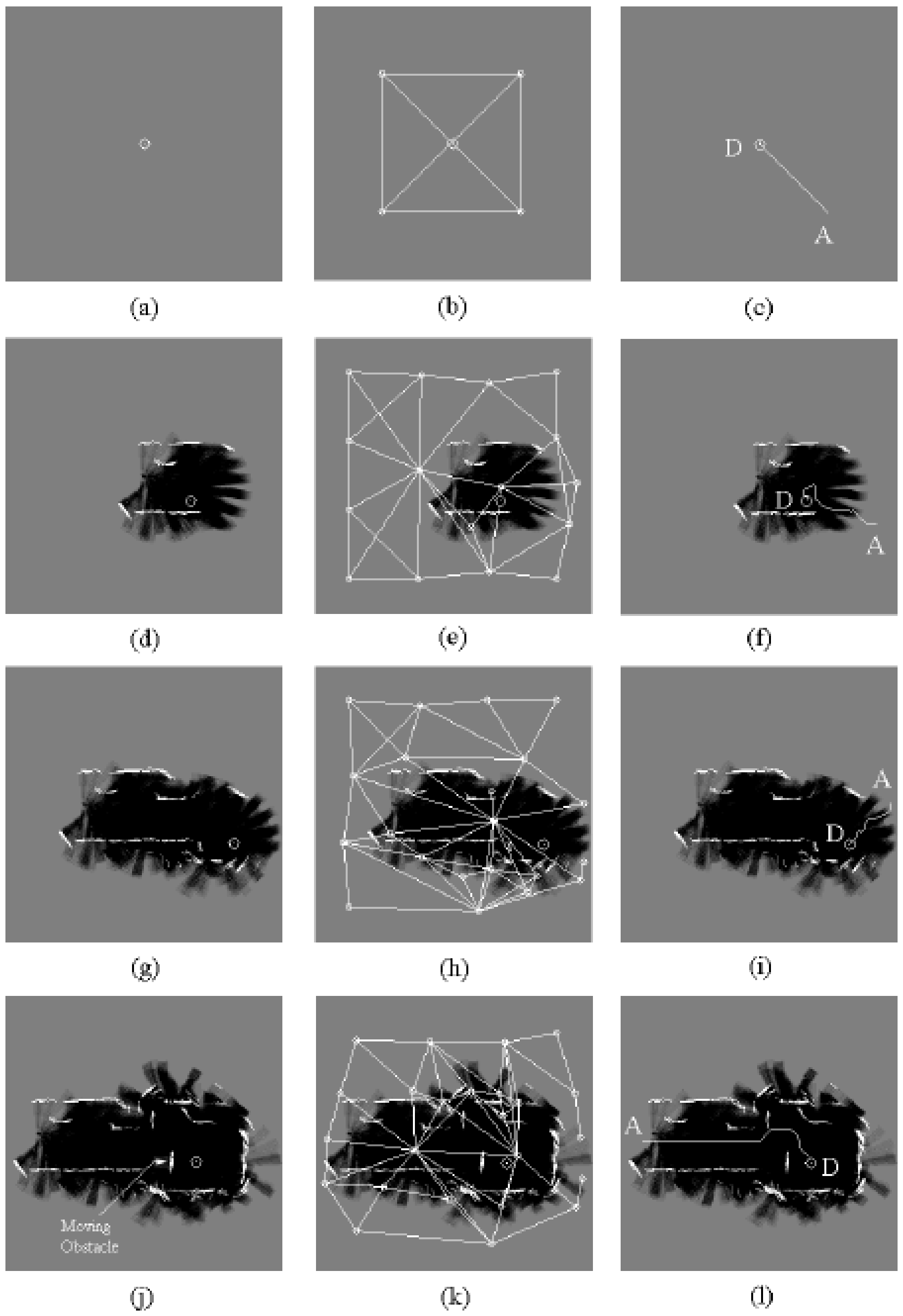

Fig. 13. Point to point navigation: metric maps, topological maps and routes proposed by the local navigator at different times.

later, as shown if Figs. 13d to i, the metric map and the topological map are updated as the robot moves, so the next node to be visited and the route to be followed changes. A mobile obstacle appears in Fig. 13j, and the path planning layer calculates a new path using the updated topological map in Fig. 13k. The final path is shown in Fig. 13l. Processing times are quite similar to the ones presented in Table 2.

6 Conclusions

This chapter has presented a hybrid architecture for sonar-based survey navigation in totally or partially unknown environments. The architecture has a client-server style. Unlike similar approaches, there is a global and several local servers rather than just one server in this style. Thus, data requests are handled faster and more efficiently. The architectural structure consists of several layers. Some of these layers are deliberative, while others are reactive. Basically, they exchange information using a hierarchical representation of the environment. The main advantages of this representation is that sensors are used to quickly and simply create and update a metric map, but high level processing is performed on a grounded topological map with a small number of nodes. Also, the whole representation can be constructed rapidly while the robot is navigating, and it explicitly represents unexplored areas at the topological level. Since deliberative planning tends to be computationally expensive, it is performed at the topological level. Reactive layers, which depend on the local geometry of the environment, work at the metric level instead. The main advantage of the proposed technique is that local navigation is modulated by deliberative planning, so that it is not only fast but efficient as well. Finally, it should be noted that layered modular systems are usually easy to expand. Our future work will focus on expanding the described architecture to add a vision module to the robot.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Technology and FEDER funds, project No. TIC2001-1758.

References

1. Dudek, G., and Jenkins, M. (2000): Computational principles of mobile robotics. Cambridge University Press, Cambridge, USA.
2. Leonard, J.J. and Durrant-Whyte, H.F. (1992): Directed Sonar Sensing for Mobile Robot Navigation. Kluwer Academic Publishers, Massachussetts.
3. Maybeck, P. S. (1990): The Kalman filter: An introduction to concepts. In I. J. Cox, & G. T. Wilfang (Eds.), Autonomous Robot Vehicles, New York: Springer, pp. 194-204.

4. Arkin, R.C. (1998): Behaviour based robotics, MIT Press, Cambridge.
5. Moravec, H.P. (1983): The Stanford Cart and the CMU Rover. Proc. of the IEEE, Vol. 71, No. 7, pp. 872-884.
6. Albus, J. (1991). Outline for a theory of intelligence. IEEE Transactions on Systems, man and Cybernetics, Vol. 3, No. 21, pp. 473-509.
7. Hu, H. and Brady, M. (1996): A parallel processing architecture for sensor based control of intelligent mobile robots. Robotics and autonomous systems, Vol. 17, 235-257.
8. Tsotsos, J.K. (1997): Intelligent control for perceptually attentive agents: The S* proposal. Robotics and autonomous systems, Vol. 21, pp. 2-21.
9. Brooks, R.A. (1991): Intelligence Without Reason. Proc. IJCAI-91', pp. 569-595, Sydney, Australia.
10. Brooks, R.A. (1986): A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, Vol. 2, No. 1, pp. 1423.
11. Lyons, D. (1992): Planning, Reactive. In Shapiro, S. (Ed.): Encyclopedia of Artificial Intelligence, 2^{nd} Edition. John Wiley & Sons, New York, pp. 1171-1182.
12. Coste-Manière, E. and Simmons, R. (2000): Architecture, the backbone of robotic systems. Proc. of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, pp. 67-72.
13. Chocon, H. (1992): Object-Oriented Design and Distributed Implementation of a Mobile Robot Control System. Proc. of Workshop on Architecture for Intelligent Control Systems, R. Chatila and S.Y. Harmon, Eds. Nice, France.
14. Wise, J.D. and Ciscon, L. (1992): TelRIP Distributed Application Environment Operating Manual, Version 1.6. Technical Report 9103, Universities Space Automation/Robotics Consortium.
15. Simmons, R.G. (1994): Structured Control for Autonomus Robots. IEEE Transactions on Robotics and Automation, Vol. 10, No. 1, pp.34-43.
16. Dulimarta, H.S. (1996): A Client/Server Control Architecture for Robot Navigation. Pattern Recognition, Vol. 29, No. 8, pp. 1259-1284.
17. Franz, M.O. and Mallot, H.A. (2000): Biomimetic robot navigation. Robotics and autonomous systems, Vol. 30, pp. 133-153.
18. Moravec, H. P. (1988): Sensor fusion in certainty grids for mobile robots. AI Magazine, Vol. 9, No. 2, pp. 61-74.
19. Matarić, M.J. (1994): Interaction and intelligent behavior. Technical Report AI-TR-1495, MIT, AI-Lab, Cambridge-USA.
20. Thrun, S., Bucken, A., Burgard, W., Fox, D., Frohlinghaus, T., Hennig, D., Hofmann, T., Krell, M., and Schimdt, T. (1998): Map learning and high-speed navigation in RHINO. MIT/AAAI Press, Cambridge.
21. Borenstein, J., Everett, H.R. and Feng, L. (1996): Navigating mobile robots: systems and techniques. Wellesley, Massachusetts: A.K. Peters, Ltd.
22. Kuipers, B.J. and Byun, Y.T. (1991): A robot exploration and mapping strategy based on a semantic hierarchy of spatial representation. Journal of Robotics and Autonomous Systems, Vol. 8, pp. 47-63.
23. Arleo, A., Millán, J.R. and Floreano, D. (1999): Efficient learning of variable-resolution cognitive maps for autonomous indoor navigation. IEEE Transactions on Robotics and Automation, Vol. 15, No. 6, pp. 990-1000.
24. Zelinsky, A. (1992): A mobile robot navigation exploration algorithm. IEEE Transactions on Robotics and Automation, Vol. 8, pp. 707-717.

25. Bandera, A., Urdiales, C. and Sandoval, F. (2001): An hierarchical approach to grid-based and topological maps integration for autonomous indoor navigation. Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001), pp. 883-888, Maui, Hawaii, USA.
26. Pagac, D., Nebot, E.M. and Durrant-White, H. (1998): An evidential approach to map-building for autonomous vehicles". IEEE Transactions on Robotics and Automation, Vol. 14, No. 4, pp. 623-629.
27. Rencken, W.D. (1993): Concurrent localisation and map building for mobile robots using ultrasonic sensors. Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 3, pp. 2192-2197, New York-USA.
28. Schiele, B. and Crowley, J. (1994): A comparision of position estimation techniques using occupancy grids. Robotics and autonomous systems, Vol. 12, pp. 163-171.
29. Chen, D., Szczerba, R. and Uhran, J. (1997): A framed-quadtree approach for determining Euclidean shortest paths in 2-D environment. IEEE Transactions on Robotics and Automation, Vol. 13, No. 5, pp. 668-680.
30. Gee, A.H., and Prager, R.W. (1995): Limitations of Neural Networks for solving the Traveling Salesman Problem. IEEE Transaction on Neural Networks, Vol. 6, No. 1, pp. 1542-1544.
31. Larrañaga, P., Kuijpers, C.M., Murga, R.H., Inza, I. and Dizdarevic, S., (1999): Genetic algorithms for the Travelling Salesman Problem: a review of representations and operators. Artificial Intelligence, Vol. 13, No. 2, pp. 129-170.
32. Latombe, J.C. (1991): Robot Motion Planning. Ed. Kluwer, Academic Publishers, Boston.
33. Koren, Y. and Borenstein, J. (1991): Potential Fields Methods and their Inherent Limitations for Mobile Robot Navigation, IEEE International Conference on Robotics and Automation, pp. 1398-1404, California, USA.
34. Ulrich, I. and Borenstein, J. (1998): VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. IEEE International Conference on Robotics and automation, pp. 1572-1577, Leuven, Belgium.
35. Minguez, J. and Montano, L. (2000): Nearness Diagram Navigation (ND): A New Real Time Collision Avoidance Approach. Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000), pp. 2094-2100, Takamatsu, Japan.