Efficient Learning of Variable-Resolution Cognitive Maps for Autonomous Indoor Navigation

Angelo Arleo, José del R. Millán, and Dario Floreano

***Abstract*—This paper presents an adaptive method that allows mobile robots to learn cognitive maps of indoor environments incrementally and on-line. Our approach models the environment by means of a variable-resolution partitioning that discretizes the world in perceptually homogeneous regions. The resulting model incorporates both a compact geometrical representation of the environment and a topological map of the spatial relationships between its obstacle-free areas. The efficiency of the learning process is based on the use of local memory-based techniques for partitioning and of active learning techniques for selecting the most appropriate region to be explored next. In addition, a feed-forward neural network is used to interpret sensor readings. We present experimental results obtained with two different mobile robots, namely a Nomad 200 and a Khepera. The current implementation of the method relies on the assumption that obstacles are parallel or perpendicular to each other. This results in variable-resolution partitionings consisting of simple rectangular partitions and reduces the complexity of treating the underlying geometrical properties.**

***Index Terms*— Autonomous mobile robots, exploration, map learning, neural networks, occupancy grid, topological graph, variable-resolution partitioning.**

I. Introduction

THIS paper presents a map learning method that allows a mobile robot to explore an unknown indoor environment in order to acquire a spatial-navigation model incrementally and on-line.

The proposed method integrates the two principal approaches to map learning for indoor environments, namely the *geometrical* paradigm and the *topological* paradigm. In the former, the geometrical features of the world are modeled accurately. Obstacles are modeled according to their absolute geometric relationships. One of the most popular of such methods consists of representing space by means of a two-dimensional evenly-spaced grid called *occupancy grid* (e.g., [1], [2]). Each grid cell estimates the occupancy probability of the corresponding area of the world. Topological maps (e.g., [3], [4]) are more qualitative representations of the world. The model consists of a graph, where nodes represent perceptually distinct regions of the world and arcs indicate spatial relations between them.

The two paradigms are characterized by complementary strengths and weaknesses [5]. Since occupancy grids reproduce the geometrical structure of the environment explicitly, they are easy to learn and maintain: the position of each observed feature is mapped into a global absolute frame of reference. As a consequence, also the robot's position and orientation within the model are automatically given by its position and orientation within the real world. This allows the robot to distinguish places of the environment that are perceptually similar. However, this approach is limited by its vulnerability to errors that affect the metric information (i.e., robot's position and distance to obstacles). In particular, failures of the robot self-localization capability have devastating effects on the map accuracy. In addition, building occupancy grids is expensive in terms of memory and time. Indeed, to accurately model each single part of a complex environment, the resolution of the occupancy grid must be high and the learner must manage a huge amount of data.

Recent research on animals' behavior suggests a more qualitative representation of the world based on a compact storage of a few relevant features of the environment [6], [7]. Links between these landmarks are then used to achieve navigation. The topological approach is inspired upon these findings. Its major advantage is the compactness of the environmental model: the complexity of the learned map is directly related to the world complexity. This permits to optimize the use of time and space resources. Furthermore, since topological maps are qualitative representations of the world, they are not necessarily vulnerable to errors in the metric information. In addition, since the world is represented by a graph, this approach permits fast planning of robot trajectories. As in the geometrical approach, robots need to use self-localization in order to build consistent representations. However, since topological maps do not rely on absolute frame of reference, the place recognition problem is solved by discriminating sensory data only. That is, landmarks (i.e., nodes in the topological graph) are identified by means of a sensory pattern recognition process. As a consequence, spatially distinct places producing equivalent sensory patterns might not be distinguished.

Our approach attempts to combine the respective advantages of the geometrical and topological paradigms and to minimize some of their weaknesses. The model consists of a *variable-resolution partitioning*: the environment is discretized in sub-

Manuscript received April 1, 1998; revised April 21, 1999. This paper was recommended for publication by Associate Editor E. Rivlin and Editor V. Lumelsky upon evaluation of the reviewers' comments. This work was supported by the Swiss National Science Foundation under Project 21-49174.96.

A. Arleo and D. Floreano are with the Microcomputing Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland.

J. del R. Millán is with the Institute for Systems, Informatics, and Safety, Joint Research Centre of the EC, Ispra (VA) 21020, Italy.

Publisher Item Identifier S 1042-296X(99)10416-6.

areas (i.e., *partitions*) having different sizes and representing perceptually homogeneous regions of the environment. The resulting map provides a compact representation of the geometrical structure of the world, thus optimizing the use of memory and of time resources. Indeed, the use of the variable-resolution directly relates model complexity to world complexity: the aim is to have a high resolution only in areas of the world that require more complex navigation. Furthermore, the map does not model fine details—which are difficult and expensive to represent. The topological aspect of the model derives from the fact that the partitioning splits the environment in perceptually homogeneous regions. Thus, it is possible to abstract a graph representing the spatial connectivity between these regions. This graph is then used for motion planning.

The proposed map learning method addresses the following issues.

1) *Model Complexity*: Instead of building a global monolithic model of the environment, we adopt a local learning approach. By means of a variable-resolution partitioning the robot's knowledge is distributed in many simple local models (e.g., [8]). Each local model represents a partition and encodes the information about the corresponding region of the world. A local approach helps for reducing the problem of *catastrophic interference* [9] that would arise in the case of using a global monolithic representation.
2) *Sensor Uncertainty*: Since perceived data are not error-free, the robot needs to interpret its sensor readings effectively. We follow Thrun's approach [5] of using a neural sensor interpretation. A feed-forward neural network is used to create a local occupancy grid modeling the space surrounding the robot. This grid provides the robot with a robust local perception. Our approach differs from Thrun's approach in the way this local grid is used. In Thrun's approach the local grid is used to build global geometrical maps directly. In our method the local grid is used to approximate obstacle boundaries by straight lines. These lines are then used to build the variable-resolution partitioning.
3) *Real-time Learning*: We take a memory-based learning approach (e.g., [10], [8], [11]) to build the map on-line and in real time. A memory-based learner is trained by simply storing data in memory, reducing the time needed to incorporate new knowledge in the model. Purely memory-based methods do not attempt any data compression (e.g., [8], [11]). The proposed method attempts to optimize resources (i.e., memory and time to manage data) by collecting only significant experiences; that is, those experiences that actually improve the model accuracy.
4) *Exploration*: In order to optimize the learning time, we use an active learning approach (e.g., [12], [13]). An "active" learner is one that uses its current knowledge to drive the generation of training data in order to maximize the gain of information in the least possible time. This active behavior is obtained by making the robot always explore the least known region of the environment.

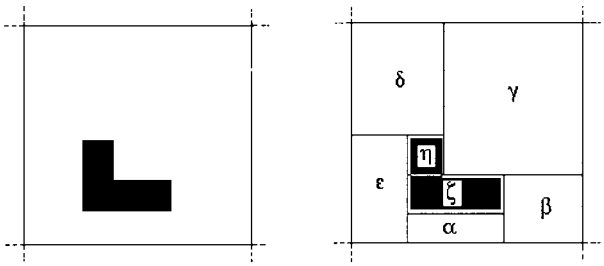
The remainder of this paper is organized as follows. Section II reviews related work. Section III describes the map learning method. Section IV shows experimental results. Finally, in Section V we discuss limitations and advantages of our approach.

II. Related Work

The proposed approach can be thought of as a combination of the geometrical and topological paradigms. The benefits of combining both approaches have been already discussed in the literature. Earliest ideas were proposed by Chatila and Laumond [14] and Elfes [15], [16]. Chatila and Laumond suggested to model objects by polyhedra and to use them to split the space into a limited number of regions corresponding to rooms, doors and so on. However, these authors only made a proposal and did not provide algorithmic details. Elfes developed algorithms for building occupancy maps using computer vision. He also suggested deriving large-scale topological models but he did not propose an algorithmic solution to this problem.

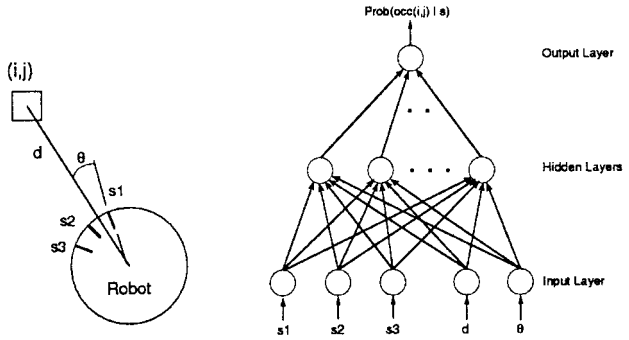
Recently, Thrun [5] has implemented a method for building large-scale metric-topological maps of indoor environments. After a global occupancy map has been learned, a topological representation is generated by splitting the metric map into a small number of coherent regions. Our method is related to Thrun's approach in several respects. The first and most obvious is the use of the neural sensor interpretation technique to provide the robot with a robust local perception. In both methods the neural sensor interpretation results in a local occupancy grid modeling the space surrounding the robot. However, the two approaches differ in the way this local grid is subsequently used. Thrun's robots utilize the local grid to acquire global geometrical maps. The neural sensor interpretation is constantly used, and subsequent local grids are integrated to form a global metric grid. In our approach, robots use the local occupancy grid only when they need to model obstacle boundaries by straight lines. Once a boundary has been approximated by a straight line, the neural sensor interpretation is not used any further and robots rely on their raw sensor readings. The second similarity between our method and Thrun's approach concerns the integration in a single method of the geometrical and topological paradigms for map learning. Nevertheless, the two approaches build quite different geometrical representations: global grid-based maps in Thrun's method versus variable-resolution partitioning in our method. Finally, our approach abstracts the topological map from the geometrical representation on-line, while Thrun's approach does it off-line and only after the whole global map is available.

The *parti-game* algorithm for the acquisition of control policies proposed by Moore and Atkeson [8] also creates a variable-resolution partitioning, but only for allowing simulated robots to learn good trajectories to goal regions. There are at least three main differences between parti-game and our method. First, since parti-game is based only on collision information, the number of partitions created for a given environment is much higher than ours. Second, one assumption



(a) (b)

Fig. 1. An example of partitioning update. (a) A partition that does not represent a homogeneous perceptual situation. (b) The resolution of the partitioning is increased to model the obstacle.



(a) (b)

Fig. 2. (a) For a given cell $(i, j) \in \mathcal{G}$, the input of the neural network $\mathcal{N}$ consists of the readings of the three sensors oriented toward the (i, j) cell $(s_1, s_2,$, and $s_3)$ and of the polar coordinates of the cell (i, j) with respect to the robot. (b) For a given cell (i, j) (identified by its polar coordinates relative to the robot) and a given sensory pattern $\mathbf{s} = (s_1, s_2, s_3)$, the output of network is interpreted as the conditional probability Prob$(occ_{ij}|\mathbf{s})$.

of parti-game is that the size of the environment has to be known. Third, parti-game has a high computational cost that may prevent on-line operation.

The proposed method also bears some similarities to Taylor and Kriegman's approach for allowing a mobile robot to explore previously unknown environments [17]. Their algorithm also models the environment by means of local representations, and it builds a relational map by recording the relationships between these local models. In their approach, however, exploration relies on vision and each local model is explicitly associated with a visually distinct and recognizable landmark. Our robots do not use vision, and the concept of landmark is just implicitly contained in the relational map derived from the variable-resolution partitioning.

III. Real-Time Map Learning

The proposed method allows a robot to model a structured indoor environment by means of a variable-resolution partitioning $\mathcal{P}$. Each partition $p \in \mathcal{P}$ is a local model that codifies the robot's knowledge about the corresponding area of the world. The aim is to create partitions that represent perceptually homogeneous areas. For instance, the partition shown in Fig. 1(a) does not meet such a requirement, because it represents both a free space and an obstacle. We call such a partition an *incoherent* partition.

An important assumption of the current implementation of our method is that all meaningful obstacle edges are aligned with either the x- or the y-axis. Such a simplified world allows the robot to create variable-resolution partitionings made by rectangular partitions only. This reduces the complexity for updating and maintaining the partitioning $\mathcal{P}$, by making the spatio-geometrical properties of the created partitions simple to treat.

The proposed method relies on a modular architecture. In particular, five principal modules can be identified.

1) *Neural Sensor Interpretation*: Sensory information is interpreted by a feed-forward neural network. This sensor interpretation results in a local occupancy grid that improves the local perception of the robot.
2) *Identifying Obstacle Boundaries*: The proposed method consists of approximating obstacle boundaries by straight lines and using them to build the partitioning $\mathcal{P}$ [see Fig. 1(b)]. This is mainly achieved by means of the neural sensor interpretation: every time the robot needs to identify an obstacle edge, it aligns with and follows that edge, and it uses the local occupancy grid to find the approximating line.
3) *Partitioning Update*: Every time the robot perceives an unknown obstacle, it increases the resolution of the partitioning $\mathcal{P}$ to include it. To do this, it approximates all the obstacle edges by straight lines. Once the whole perimeter of the obstacle has been modeled, the resolution of $\mathcal{P}$ is increased in the local area containing the obstacle—with the new partitions modeling the obstacle.
4) *Exploring the Environment*: The robot is always exploring the environment to improve its current map. Given the current partitioning $\mathcal{P}$, the *explorer* module selects as target the partition corresponding to the least known region of the environment.
5) *Planning and Action*: Given the target partition selected by the explorer, the *planner* computes the optimal path across $\mathcal{P}$. Then, low-level controllers actually bring the robot there.

The above modules are described in more detail in the following sections.

A. Neural Sensor Interpretation

A proper sensor interpretation is useful because sensor readings are typically corrupted by noise whose distribution is generally unknown. Another concern is the robot's requirement to interpret all its sensor readings simultaneously. To address these issues we follow the approach proposed by Thrun [5] where a feed-forward neural network $\mathcal{N}$ builds a local occupancy grid $\mathcal{G}$ from sensor readings. The local occupancy grid, which consists of $n \times n$ cells, is in fact a local view that moves and rotates with the robot. The choice of the size and the resolution of the grid $\mathcal{G}$ is given by a trade-off between computational cost and reliability of edge detection.

For a given cell $(i, j) \in \mathcal{G}$, the input of $\mathcal{N}$ consists of (Fig. 2)

1) the readings $\mathbf{s} = (s_1, s_2, s_3)$ of the three sensors oriented in direction of (i, j);

2) the polar coordinates θ_{ij} and d_{ij} of the center of the cell (i,j) with respect to the robot (the angle θ is calculated with respect to the first of the three closest sensors).

For each cell $(i,j) \in \mathcal{G}$, the network $\mathcal{N}$ produces a value Prob$(occ_{ij}|\mathbf{s})$ that measures the probability of this cell being occupied given the sensory pattern $\mathbf{s} = (s_1, s_2, s_3)$.

It is worth noting that once such a network has been trained, it can be used by the robot in several differently structured indoor environments. Another important feature is that this sensor interpretation strategy is *platform-independent*, in the sense that it can be applied on mobile robots having different sensor configurations.

1) Training the Neural Network: Training data are of the form $\langle s_1, s_2, s_3, \theta_{ij}, d_{ij}, o_{ij} \rangle$, where o_{ij} is the desired output. They are generated by randomly placing and orienting the robot in a known environment (i.e., an environment where the position of obstacles is known). For each position and orientation of the robot, the elements (i,j) of the local grid are randomly sampled. The target output o_{ij} is the true occupancy state of (i,j) computed by considering the intersection between the cell (i,j) and the known obstacles.

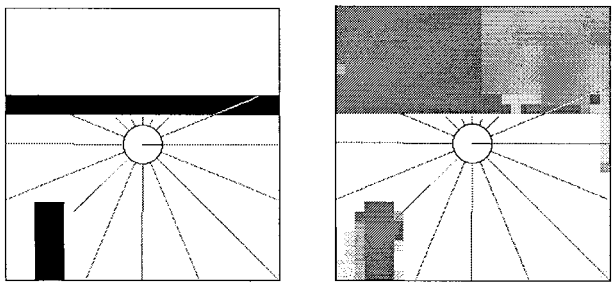
The neural network $\mathcal{N}$ is trained off-line by a gradient descent method [18]. Input and output values are normalized in the range [0,1]. As stated before, the network $\mathcal{N}$ produces the conditional probability Prob$(occ_{ij}|\mathbf{s})$ given the sensory pattern $\mathbf{s} = (s_1, s_2, s_3)$ [5]. It has been shown that the output of a neural network can be interpreted as a posterior probability if the network is trained by minimizing the cross-entropy error function [19]. We trained the network $\mathcal{N}$ by using back-propagation with momentum term [18] to minimize the cross-entropy error function E

$$E = -\sum_n \{o_{ij}^n \ \ln \ y^n + (1 - o_{ij}^n) \ \ln (1 - y^n)\}$$

where Σ_n sums over all training examples, and o_{ij}^n and y^n are the target and the actual output associated to the n_{th} example, respectively.

We adopt an adaptive learning rate η. Initially $\eta = \eta_{\max}$ and then its value decreases linearly until it reaches a sufficiently small value $\eta_{\min}$. If, at this point, the error E is still higher than desired, then the value of η is restored to $\eta_{\max}$ and the process is repeated again. This cycle is iterated until the performance of the network does not improve significantly. The rationale behind this technique is that when η is increased, the weight configuration is perturbed so as to climb the walls of the basin of attraction it was in.

The architecture of $\mathcal{N}$ is built incrementally [20]. Initially, the network has just one hidden unit and, as learning proceeds, a new hidden unit is added when the current network cannot reduce the error E any further. The new unit is added either in an existent hidden layer or in a new one. By modifying the network architecture, the shape of the weight space is also changed, which might remove the local minimum where the network is trapped. Our experimental evidence shows that when re-training with a new hidden unit the performance of the network improves. This idea can help to build a "minimum-size" network to solve the task at hand. In preliminary explorations we observed that the performance of the incrementally-built network is better than the performance of a network trained from the beginning with the same final architecture.



(a) (b)

Fig. 3. An example of neural sensor interpretation. (a) A part of the environment and the robot within it. The robot is following the horizontal wall to approximate it by a straight line. The snapshot has the same area of the local grid $\mathcal{G}$. The robot sensor readings (sonars in this case) are represented by radial lines. (b) The corresponding local grid obtained by using the network $\mathcal{N}$ and integration over time. The darker a cell, the higher its probability of being occupied.

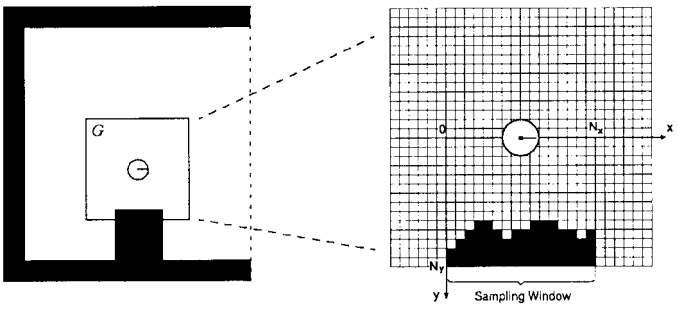
2) Integration over Time of Occupancy Probabilities: As previously mentioned, the robot is using the network $\mathcal{N}$ as it follows obstacle boundaries. Then, consecutive neural sensor interpretations can be integrated over time in order to obtain a more reliable local occupancy grid $\mathcal{G}$. If, for a given sensory pattern at time t, $\mathbf{s}^t$, the output of $\mathcal{N}$ is the conditional probability Prob$(occ_{ij}|\mathbf{s}^t)$, then, given N consecutive sensor readings $\mathbf{s}^1, \mathbf{s}^2, \cdots, \mathbf{s}^N$ the occupancy probability for the cell (i,j) can be thought as Prob$(occ_{ij}|\mathbf{s}^1, \mathbf{s}^2, \cdots, \mathbf{s}^N)$. Time integration is achieved by applying Bayes' rule for estimating this probability [1], [5]

$$\text{Prob}(occ_{ij}|\mathbf{s}^1, \mathbf{s}^2, \cdots, \mathbf{s}^N) = 1 - \left(1 + \prod_{n=1}^{N} \frac{\text{Prob}(occ_{ij}|\mathbf{s}^n)}{1 - \text{Prob}(occ_{ij}|\mathbf{s}^n)}\right)^{-1}.$$

Fig. 3 shows an example of a local grid built by using the network $\mathcal{N}$ to interpret the sensor readings, and by using Bayes' rule to integrate over time. This example highlights some benefits of this approach. For instance, it illustrates how the neural interpretation can compensate errors due to specular reflections (e.g., the white ray entering the wall, in the upper right corner). In addition, notice the obstacle in the lower left corner of Fig. 3(a): the integration over time of consecutive interpretations produces an acceptable "reconstruction" [Fig. 3(b)] despite the limited current sensory information (just one of the sensors is currently detecting the obstacle).

B. Modeling Obstacle Boundaries

The use of a neural network to interpret sensor data has been originally proposed by Thrun [5] to acquire global geometrical maps. In our approach, however, the robot uses the grid $\mathcal{G}$ only to identify the boundary of the obstacle it is aligned with and trace a line (Fig. 3). A boundary consists of those cells (i,j) with Prob$(occ_{ij}) > 1 - \epsilon$ that are closest to the robot. The parameter ϵ permits to define the threshold above which the



(a) (b)

Fig. 4. (a) The robot uses the grid $\mathcal{G}$ to identify the boundary of the obstacle it is aligned with. (b) A boundary consists of cells (i,j) with Prob$(occ_{ij}) > 1-\epsilon$ that are closest to the robot. To determine a straight line that goes through these cells we use the χ^2 method.

occupancy probability provided by the neural network codes for an obstacle. The problem, then, is to find a straight line that goes through these cells.

Consider the local grid at time t depicted in Fig. 4. A sampling window [Fig. 4(b)] defines the width of the set of cells to be fitted by a straight line. Each cell (i,j) in the sampling window is thought as a point (x,y) $(x = 1 \cdots N_x$, $y = 1 \cdots N_y)$. Now, consider, for each $x_n \leq N_x$, the minimum y_n for which the relationship Prob$(occ_{x_n y_n}) > 1-\epsilon$ holds. This yields a set of N_x points (x_n, y_n). Then, the problem is to find the best straight line

$$y(x) = y(x;a,b) = a + bx$$

approximating this set of points. To determine this line we use a simple version of the χ^2 method [21].

As previously mentioned, the straight lines modeling the obstacle edges are used to build the partitioning $\mathcal{P}$ [see Fig. 1(b)]. In order to keep the complexity of $\mathcal{P}$ as low as possible, the robot adopts a simple strategy. Every time it finds the straight line l_1 modeling the current obstacle boundary, it checks whether l_1 might be aligned with any of the existing lines. If there is a line l_2 oriented as l_1, and the orthogonal distance d between l_1 and l_2 is smaller than a threshold d_ϵ, then the robot makes the line l_1 to be aligned with l_2. This simple technique also permits to create a map that fits structural regularities of the environment (e.g., the two in-line walls holding a door frame).

The accuracy of $\mathcal{P}$ depends on two factors, namely the resolution of the local grid and the accuracy of the straight lines approximating the obstacles. The combined effect of the neural sensor interpretation and of the line fitting method is such that fine details of the environment are not included in the partitioning process. Indeed, the aim is to model the spatial structure of the world qualitatively. Fine details are not necessary for high-level planning and they can be handled by a low-level reactive module.

C. Partitioning Update

Every time the robot perceives an unknown obstacle it decides to increase the resolution of the partitioning $\mathcal{P}$ to model it. The robot approaches the obstacle and aligns with one of its boundaries. Then, it follows that boundary and starts using the local grid $\mathcal{G}$ and the χ^2 method. As soon as the robot finds the straight line that approximates this first boundary, it stops utilizing the local grid $\mathcal{G}$, that is, the neural sensor interpretation. Then, it moves along that line until the end of the boundary utilizing the raw sensor readings only. Then, the robot rotates to align with the new boundary and starts modeling it by following the edge and using the local grid $\mathcal{G}$ and the χ^2 method. Again, as soon as the approximating line is found, the robot stops using the neural sensor interpretation. At this point, it computes the intersection between the current straight line and the previous one to identify the corner previously visited. Then, the robot moves until the end of the boundary using the raw sensor readings only, rotates to align with the new boundary and repeats the process. This process stops when the robot has modeled the whole obstacle (i.e., when it has traveled around the whole perimeter of the obstacle). This stop condition is recognized by taking into account the two straight lines modeling the first and the second obstacle boundaries. Every time the robot reaches the end of a boundary, it checks if it is meeting the line modeling the first obstacle edge. If this is the case, it rotates to align with the new boundary, it models it and it goes to the end of the edge. If there it meets the line approximating the second obstacle edge, it considers the whole obstacle as modeled. Notice that this procedure results in recognizing the first corner created during the obstacle modeling process. However, the fact of looking for lines instead of points (corners) provides a more robust corner identification process despite the odometry error.

Once all corners of an obstacle have been memorized, it is possible to increase the resolution of the partitioning $\mathcal{P}$ to model the new obstacle: each new corner is connected to the closest perpendicular edge of one of the existing partitions [see Fig. 1(b)]. This strategy always creates rectangular partitions. It is worth noting that the resolution of $\mathcal{P}$ is increased only in local areas containing unknown obstacles. This meets our aim of having a higher map resolution only in critical areas of the environment.

Two adjacent partitions are considered redundant if both represent either obstacle or free space and they can be merged to produce a rectangular partition. It might happen that increasing the resolution generates a redundancy in the partitioning $\mathcal{P}$. After updating $\mathcal{P}$, redundant partitions are removed over all $\mathcal{P}$. Thus the resulting model is as compact as possible and its computational complexity is kept low.

After updating the partitioning $\mathcal{P}$, the robot stores the knowledge concerning physical transitions between new partitions. In particular, the robot keeps a database D of negative experiences memorizing the "forbidden transitions" between the new partitions containing the obstacle and the adjacent partitions. A negative experience between a partition α and a partition β is stored as a triplet of the form $(\alpha, \beta, false)$. In the example of Fig. 1(b), the database D would be updated as follows: $D = D \cup \{(\alpha,\zeta,false),(\beta,\zeta,false),\ (\gamma,\zeta,false),(\gamma,\eta,false),$ $(\delta,\eta,false),\ (\epsilon,\eta,false),(\epsilon,\zeta,false)\}$. The database D is the robot *long-term memory* of the spatial relationships between partitions and it is used by the planner (Section III-E) to derive a topological graph from the partitioning $\mathcal{P}$.

D. Exploring the Environment

An important concern of the proposed method is to devise an efficient exploration of the environment in order to accelerate the map learning process.

Since the environment is initially unknown, the robot begins with an empty partitioning (i.e., $\mathcal{P} = \emptyset$). At this time, it starts exploring by moving along a straight trajectory in a randomly chosen direction. As soon as it detects an object, it starts creating the variable-resolution partitioning to model it. When $\mathcal{P} \neq \emptyset$, exploration is achieved by selecting a target partition, moving across the environment to reach it, and, finally, exploring the target partition exhaustively. As soon as the robot detects an unknown object (either within the target or along the path leading to it), it gives up exploration and starts updating the partitioning $\mathcal{P}$. When the unknown object has been modeled, exploration is resumed by selecting a new target partition.

The *explorer* is the module responsible for the choice of the target partition. We adopt an active learning technique [12]: the robot selects the next region to be explored so that it obtains an environmental map with the smallest possible number of exploration steps. This active behavior is obtained by making the robot always explore the least known region of the environment.

Active exploration is based on estimating the exploration utility U_e of each existing partition and selecting the one which maximizes it [13], [22]. This heuristic real-valued function measures "how much a partition is worth to be explored." To define U_e, we use a technique called *counter-based exploration with decay* [13]. It gives higher utility to partitions that have been visited less often and less recently. A counter $c(p)$ keeps track of the number of occurrences for each partition $p \in \mathcal{P}$ (i.e., how many times that partition has been visited). In order to take into account when a partition has been visited, the counter $c(p)$ is multiplied by a decay factor $\lambda \leq 1$. Thus, whenever the explorer module is triggered to select a new target partition, it updates the exploration utilities as follows:

$$c(p) = \lambda \cdot c(p) \quad \forall p \in \mathcal{P}$$

$$U_e(p) = \begin{cases} 1/c(p) & c(p) > 0 \\ K & c(p) = 0 \end{cases}$$

where $K > 1$ is a constant factor.

Finally, the explorer selects as target the partition that maximizes U_e:

$$target = \arg \max_{p \in \mathcal{P}} U_e(p).$$

E. Planning and Action

Given the target partition, the robot invokes the *planner* to compute a trajectory toward it. The planner derives a topological graph from the current partitioning $\mathcal{P}$ where

1) nodes correspond to partitions;
2) arcs are derived from the long-term memory of experiences D. The node corresponding to the partition α is connected to the node corresponding to the adjacent partition β if $(\alpha, \beta, false) \notin D$, that is the transition $\alpha \rightarrow \beta$ is not characterized by a negative experience.

LOOP

1 *Exploration*: Selection of the next target; i.e., the partition with the highest U_e.
2 *Planning*: Computation of a trajectory to the target through the current partitioning.
3 *Action*: The robot actually moves controlled by the reactive module.
4 **If** the robot reaches the target, **then** it explores it exhaustively.
5 *Map Update*: **If** an incoherency has been detected, **then** the robot gives up exploration and models the unknown obstacle.
  5.1 The robot approaches the unknown obstacle.
  5.2 **While** the robot does not visit a known corner
    5.2.1 The robot aligns with one of the boundaries of the obstacle.
    5.2.2 *Neural Sensor Interpretation + Integration over time + χ^2 method*: The robot follows the boundary and approximates it by a straight line.
    5.2.3 The robot moves until the end of the boundary.
    5.2.4 The robot memorizes the corner.
  5.3 *Increase the resolution of the partitioning.*
  5.4 *Update the database of negative experiences D.*
  5.5 *Remove redundant partitions over all $\mathcal{P}$.*
  5.6 *Abstract a new topological graph.*
goto LOOP

Fig. 5. The map learning method.

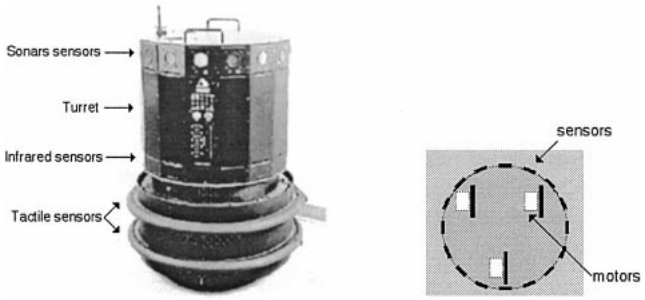
Let P the number of partitions forming the variable-resolution partitioning $\mathcal{P}$. Notice that the number of nodes in the graph is always smaller or equal to P, because only partitions representing obstacle-free areas are actually mapped into nodes. Then, starting from the node corresponding to the current partition, the planner searches the graph for the shortest path to the node associated with the target. Due to the low complexity of the learned partitioning $\mathcal{P}$, this process is not expensive.

Once the optimal path has been determined, a low-level planner computes the robot's trajectories between adjacent partitions in the path. In the current implementation the robot always follows straight trajectories parallel to the x and y axes of the environment. This simple motion strategy minimizes errors in the dead-reckoning system. If the robot has to move from the partition i to the adjacent partition j, and l is the boundary between them, the robot first moves parallel to l until it is in front of its middle point. Then, it moves perpendicular to l until it crosses the boundary.

Finally, a reactive low-level module controls the robot displacements by handling small inconsistencies of the map (i.e., fine details not being modeled) and possible moving obstacles (e.g., people). When modeling new obstacles, this same reactive module makes the robot follow the obstacle boundaries.

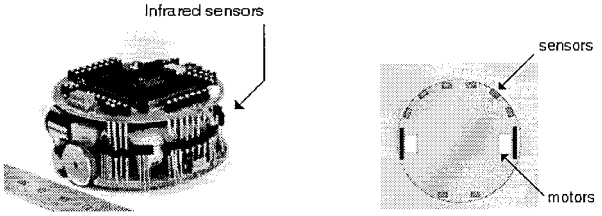
F. Algorithm Overview

The proposed map learning method results in a cyclical process dominated by exploration and model update activities. The robot is continuously exploring the environment, and exploration is driven by the acquired knowledge. The robot only gives up exploration to incorporate unknown obstacles in the model by updating the partitioning resolution. An outline of the whole algorithm is given in Fig. 5.



(a) (b)

Fig. 6. (a) TESEO, a Nomad 200 mobile robot. It is equipped with a ring of 16 sonar sensors on the top, a ring of 16 infrared sensors in the middle and 20 tactile sensors at the bottom. Each sensor ring covers 360°. (b) Schematic top-view of the robot.



(a) (b)

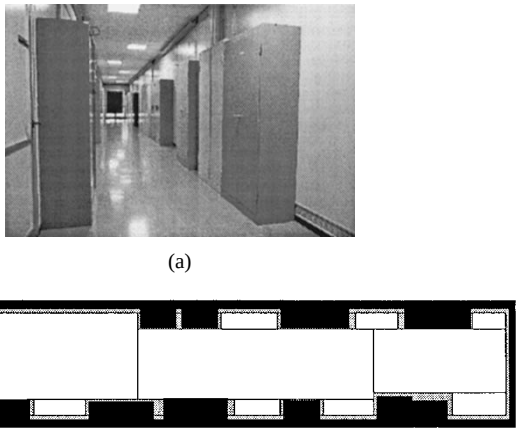
Fig. 7. (a) The Khepera mobile miniature robot. It has eight infrared sensors: Six sensors cover the frontal 180° of the robot and two sensors face backward. (b) Schematic top-view of the robot.

IV. Experimental Results

In this section we present experimental results obtained with two quite different mobile robots, namely a Nomad 200 (Fig. 6) and a Khepera (Fig. 7).

The Nomad 200 [Fig. 6(a)] has a diameter of about 50 cm and it is about 80 cm tall. This robot is equipped with 16 ultrasonic (sonar) sensors and 16 infrared sensors providing distance information from objects and 20 tactile sensors detecting collisions. Sonars and infrareds are evenly placed around the perimeter of the turret, whereas the tactile sensors cover all the perimeter of the robot below the turret. Sonar sensors can detect objects located at a distance between 15 cm and 6.5 m, whereas infrared sensors have a maximum range of about 40 cm. Each sensor ring ensures a 360° coverage [Fig. 6(b)]. Finally, a dead-reckoning system permits self-localization by keeping track of the robot position and orientation. The Nomad 200 has three independent motors. The first motor moves the three wheels of the robot together, the second one steers the wheels together and the third motor rotates the turret.

The Khepera platform [Fig. 7(a)] is a miniature robot having a diameter of about 5.6 cm. In the basic configuration used here it is 3.6 cm tall. A set of eight infrared sensors allows the robot to perceive objects within a maximum range of about 4 cm. Six of the infrared sensors are covering the frontal 180° of the robot while the remaining two sensors cover approximately 100° on the back side [Fig. 7(b)]. A dead-reckoning system is used for the auto-localization task. Finally, two motors move the two robot wheels independently.



(a) (b)

Fig. 8. (a) A corridor of our building used as testing environment. (b) The corresponding learned map (the right side of the map corresponds to the end of the corridor). Black rectangles represent obstacles (i.e., walls and cabinets), thin lines partitions. The black circle represents the robot at the initial location. Grey partitions represent areas of the world classified as obstacles by the robot.

The differences between the two robots, especially their sensor capabilities and sensor configurations, made it interesting to test our method on both of them.

A. Experiments with the Nomad 200

Fig. 8 shows one of the corridors of our building and the corresponding learned map. The variable-resolution partitioning consists of only 21 partitions. The portion of the corridor modeled measures about 20×2.5 m. The map was built with all doors closed. For the neural sensor interpretation, we use a local grid $\mathcal{G}$ of 28×28 cells, each covering an area of about 13×13 cm. The robot occupies the 4×4 central cells. Notice how the map does not represent fine details such as small gaps between cabinets. Also, the map has captured the main environmental regularities. This is mainly due to the alignment procedure that makes a new boundary line move if there exists already another line that is sufficiently close and has the same orientation.

In order to illustrate the performance of our approach in a more complex environment, we also report results obtained in simulation [23]. The Nomad 200 simulator models the robot's motion system (i.e., translation, steering, and turret rotation), and the robot's sensor system (i.e., tactile, infrared, and sonar) adequately. Uncertainty of the motion control is modeled by keeping track of two positions of the simulated robot, namely the encoder position and the actual position. The encoder position is calculated by odometry from the ideal commanded velocities, while the actual position is calculated from perturbed velocities. The perturbation on commanded velocities cares for the uncertainty in the control system, modeling robot's drift and slippage. Sensor data, which are computed based on the actual position of the robot, are modeled by considering the presence of noise as well. Both models, for motion and sensor readings, rely on a set of parameters that can be adjusted to fit real data.

Fig. 9 shows the simulated world used for carrying out the experiments. It simulates a real environment of about

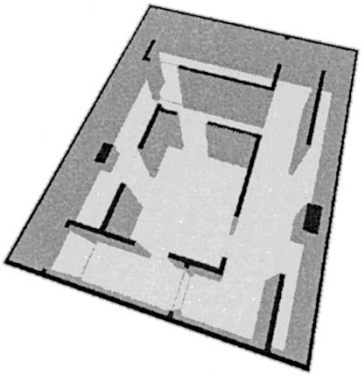


Fig. 9. The environment used for the Nomad 200 simulator. It simulates a real indoor environment of about 14.5×10.5 meters.

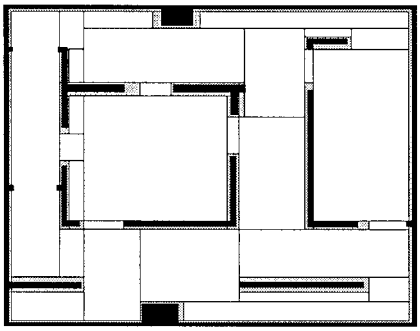
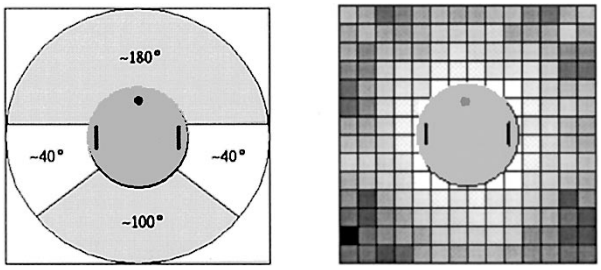


Fig. 10. The environment and the final map. Black rectangles represent obstacles. Grey partitions represent areas of the world classified as obstacles by the robot.

14.5×10.5 m. The neural sensor interpretation uses the same local grid $\mathcal{G}$ as the real robot. Fig. 10 shows the environment as well as the final map. Qualitatively, the partitioning models the free space quite accurately. Even if the map does not model some small protuberances of the obstacles properly, these "inconsistencies" do not harm the robot performance since they are useless at planning time and are handled by the reactive module. The learned map has a small number of variable-resolution partitions $(P = 45)$, which permits a fast planning of robot trajectories. As discussed in Section III-B, every time the robot models a boundary by a straight line, it also checks whether the new line can be aligned with an existing one. This strategy results in a map that preserves environmental regularities. This can be observed in Fig. 10 considering partitions modeling doorways and door frames. In addition, since new lines are aligned to previously created lines, this technique permits to create a map that partially compensates for cumulative dead-reckoning errors.

Since partitions are implicitly labeled as either "occupied" or "free" by the robot, a simple way of measuring quantitatively the accuracy of the learned map is to compute the misclassified fraction of the total area of the world. Let $\mathcal{A}_e$ be the sum of the surface of misclassified partitions (free ones classified as occupied and occupied ones classified as free) and let $\mathcal{A}_{tot}$ be the total surface of the environment. Now, the error e is:

$$e = \frac{\mathcal{A}_e}{\mathcal{A}_{tot}}. \tag{1}$$



(a) (b)

Fig. 11. (a) Effective infrared sensor coverage. Notice the two 40° blind spots on the left and right sides. (b) Average error distribution obtained by evaluating the trained network on a test set of 5000 patterns. The darker a cell, the higher the corresponding average error.

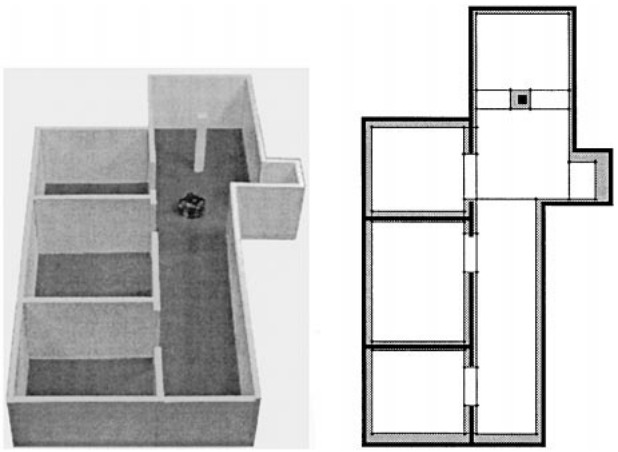
For the map of Fig. 8(b) $e = 0.0737$, whereas for the final map of Fig. 10 $e = 0.0676$.

B. Experiments with the Real Robot Khepera

In this section we present results obtained with a real mobile robot Khepera. The local grid $\mathcal{G}$ has 14×14 cells, each covering an area of 1×1 cm. The robot occupies the 6×6 central cells.

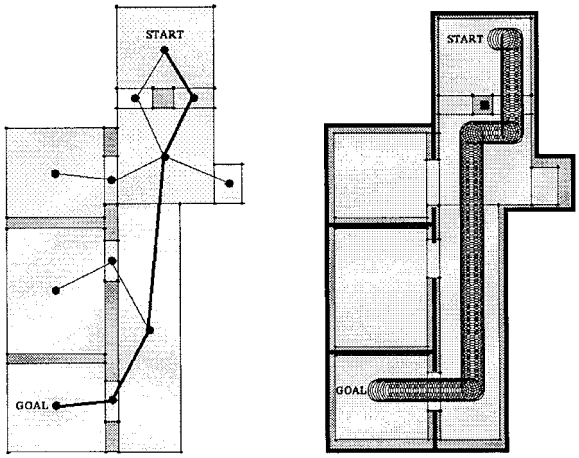
As already mentioned, the Khepera sensor configuration does not provide a 360-degree coverage [Fig. 11(a)]. Moreover, on-board infrared sensors can only detect objects which are very close. Fig. 11(b) shows the error distribution over the local grid surface obtained by evaluating the performance of the trained network on a test set of 5000 patterns. The darker a cell, the higher the corresponding mean error. Fig. 11(b) shows that the sensor interpretation provided by the trained network helps to partially compensate for incomplete sensor coverage. This extrapolation capability of the neural network is due to the fact that the obstacles used to train the network are walls, whose length scale is typically larger than the 40° blind spot. Notice that, due to the short-range response of the sensors, the performance of the network is worse in the proximity of the corners of the grid. However, while the robot is modeling an object boundary, the integration over time of consecutive neural sensor interpretations (see Section III-A2) helps to reduce this effect.

In order to improve the self-localization capabilities of Khepera we resort to off-line techniques for measuring and correcting systematic odometry errors. We use the experimental procedure *UMBmark* [24] to calibrate the two dominant systematic error sources, namely *unequal wheel diameters* (E_d) and the *uncertainty about the effective wheelbase* (E_b). This technique measures errors E_d and E_b quantitatively and then derives the compensation factors to be included in the control software. We also apply another simple procedure for odometry calibration. We make a set of n test runs in which the robot moves straight for a given distance d. For each run i, the offset ϵ_i between the asked nominal distance d and the actual distance $\tilde{d}$ is observed. Then, the expected uncertainty offset $\hat{\epsilon}$ is estimated by averaging over all test runs; i.e., $\hat{\epsilon} = (1/n)\ \Sigma_{i=1}^{n}\ \epsilon_i$. $\hat{\epsilon}$ is used in both the control software and in the odometry computation to compensate the observed offset.



(a) (b)

Fig. 12. (a) The environment used for the Khepera and the robot within it. It is a scaled (about 1:14) wood model of our laboratory covering a surface of about 100×60 cm. (b) The learned variable-resolution partitioning. Grey partitions represent areas of the world classified as obstacles, whereas the black rectangles represent the actual obstacles.



(a) (b)

Fig. 13. (a) Topological graph corresponding to the map of Fig. 12(b). (b) Robot's trajectory to go from START to GOAL. Given the path highlighted in (a), the *low-level planner* is used to compute the actual trajectories between adjacent partitions, and the reactive controller brings the robot along them. Trajectory plots are built using calibrated odometry readings from the physical robot.

Fig. 12(a) shows the real testing environment. The environment extends over a surface of about 100×60 cm. Fig. 12(b) shows both a two-dimensional representation of the world and the learned map. The resulting variable-resolution partitioning has a very small number of partitions ($P = 19$). The low complexity of the model matches the low complexity of the geometrical structure of the real environment. For the map shown in Fig. 12(b) the fraction of the total area of the environment which has been misclassified (1) is $e = 0.1275$.

Again, the alignment technique discussed in Section III-B helps to exploit the high regularity of the environmental structure and to partially compensate for odometry errors. This results in a very regular partitioning.

Fig. 13(a) shows the topological graph corresponding to the acquired partitioning. The low complexity of the topological graph is a consequence of maintaining as compact as possible the partitioning $\mathcal{P}$. Then, given a target region, planning a path leading to it is an inexpensive process. Consider the path selected by the planner to go from position START to the position GOAL. Given this path, the low-level planner computes the actual trajectories between adjacent partitions and the reactive controller takes the robot along them (Section III-E). Fig. 13(b) shows the trajectory effectively followed by the robot. As mentioned in Section III, the robot always moves parallelly or orthogonally to the x and y axes of the environment in order to reduce wheel slippage and drift.

As described in Section III-D, the robot tries to optimize its exploration process by always visiting those partitions that maximize the exploration utility U_e. In order to evaluate the exploration process, we define the following utility function:

$$U_m(t) = \frac{\sum_{p \in \mathcal{P}(t)} U_e(p,t)}{P(t)}$$

where $P(t)$ is the number of partitions of $\mathcal{P}$ at time t. $U_m(t)$ gives the global mean exploration utility corresponding to the partitioning $\mathcal{P}$ after t exploring trajectories (i.e., after the robot has run t times the map learning algorithm of Fig. 5). The lower the value of U_m, the better the environment is explored. Fig. 14 compares the performances of the used active exploration and of a random exploration during the map learning process (of the environment of Fig. 12). U_m is kept equal to 1 until at least one partition has not yet been visited. The diagram shows that active exploration overcomes random walk. The benefit of choosing carefully the next target partition is higher in the first part of the learning process. The small cardinality of the successive partitionings increases the probability of randomly selecting the most useful partition to be explored next. Furthermore, as the environment becomes uniformly explored, the standard deviation of U_m decreases and whichever partition is chosen for exploration yields the same information. Therefore, the performance difference between the two strategies is reduced. This residual difference is

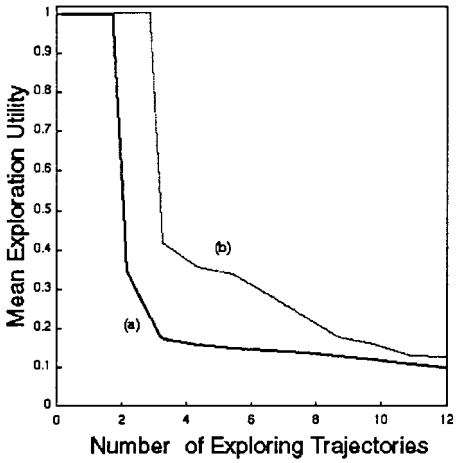


Fig. 14. Exploration performance during the learning process. The diagram shows the global mean exploration utility curves of our active exploration method (a) and of a random exploration (b). Active exploration overcomes random walk mostly in the first part of learning. Then, once the standard deviation of $U_m(t)$ decreases, random walk reports a closer performance.

proportional to the complexity of the environment. Thus both strategies achieve similar performances after a while.

V. Discussion

The proposed map building method attempts to achieve a good trade-off between representation accuracy and learning efficiency. Indeed, using variable-resolution representation relates the model complexity to the complexity of the environment: the aim is to have a high resolution only in critical areas (e.g., around obstacles) even if the map does not model fine details. The low complexity of the learned variable-resolution partitioning is a crucial feature for saving memory and time resources. If, for instance, we applied a standard geometrical approach to learn a global grid-based map of the environment shown in Fig. 12(a), then an appropriate grid resolution would consist of cells of 1×1 cm[1]. In this case, the two-dimensional evenly-spaced grid would consist of about 6000 cells. The computational complexity of managing such a map does not reflect the actual geometrical complexity of the physical environment. Having a fine resolution to represent large free areas of the world is not appropriate. Another important concern is the high cost of planning trajectories in such a representation. Searching an optimal path in a space of 6000 states is not a trivial and cheap process. On other hand, the map shown in Fig. 12(b) consists of just 19 partitions. This requires a small amount of memory for storing the adjacency structure representing the partitioning, and of time for managing it. Moreover, the simplicity of the topological graph derived from such a partitioning permits inexpensive path planning.

Computational efficiency is further improved by a limited use of the neural sensor interpretation—and, consequently, the integration over time. Building a $n \times n$ local grid $\mathcal{G}$ (Section III-A) requires $n \times n$ activations of the neural network in order to compute the occupancy probability of every cell. In our method, as mentioned in Section III, the robot uses $\mathcal{G}$ only to approximate the boundary of the obstacle it is following, a process that generally takes only a few steps. Once the robot has found a straight line approximating the boundary, it stops computing $\mathcal{G}$ and relies on its raw sensor readings to reach the end of the boundary.

An important assumption of the current implementation of our method is that all the obstacles are parallel or perpendicular to each other. This orthogonality assumption results in simple variable-resolution partitionings consisting of rectangular partitions only. This reduces the complexity of treating the underlying geometrical properties (e.g., mutual spatial relationships between adjacent partitions). This assumption also allows the robot to move along simple trajectories, which reduces the risk of wheel slippage and drift. In order for the method to deal with more general environments, this assumption must be removed. In this case, the variable-resolution partitioning would consist of polygonal partitions, which would make it more difficult to update and maintain the model.

[1]Given the Khepera features, grids with a lower resolution could fail to model the geometrical structure of the world accurately (see Section IV-B).

A critical aspect of our approach is that it relies on good robot self-localization capabilities. Currently, position and orientation of the robot are determined by dead-reckoning. We have incorporated some strategies to increase the odometry accuracy, namely off-line compensation of systematic errors (Section IV-B) and generation of straight trajectories (Section III-E), that have proven sufficient during all experiments carry out so far. However, robots must be endowed with on-line calibration techniques if we want them to work for longer periods of time and to follow more flexible trajectories. To this end, we are incorporating two techniques to the current approach. The first consists of using known corners as calibration points for the robot. During map building the robot memorizes the relative position of the corners in every partition. Then, every time it goes through a known corner, it calibrates its odometry system with the position of that corner. The second technique is based on the idea of using the neural sensor interpretation to learn a local model of every transition between adjacent partitions. Then, whenever the robot is in the vicinity of the boundaries of one of the partitions, it uses the correlation of the local grid $\mathcal{G}$ (constructed on-line as the robot moves) with the learned local model to correct possible errors of the dead-reckoning system [25], [26].

The environments used so far for evaluating our method were all static. The proposed approach could be extended to deal with dynamic environments by modifying the local representation of areas of the environment that have changed. The use of a decay factor make the explorer drive the robot to regions not recently visited, and thus the robot might detect dynamic regularities such as doors. As a consequence, it might modify the model to incorporate changes such as open (closed) doors that were previously closed (open). However, the current implementation of our approach does not model this kind of partitions as "dynamic;" i.e., it does not label them as regions whose occupancy could change over time.

A limitation of our map learning approach is its inappropriateness for small, cluttered areas. But in this kind of areas reactive strategies have demonstrated their robustness and efficiency, especially when learning is involved (e.g., [10]). We plan to combine these two complementary approaches (i.e., map learning and reactive learning) in a single navigation architecture. If the robot enters a room that it cannot model, it might resort to reactive strategies. It is worth noting that once the environment has been partitioned into a topological map, the robot must only learn efficient sensor-based strategies to move from a given node to the neighboring ones. Thus the acquired sensory-motor rules are goal-independent.

Acknowledgment

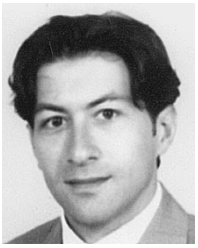
The authors would like to thank W. Gerstner for useful criticism.

References

[1] H. P. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI Mag.*, vol. 9, pp. 61–74, 1988.

[2] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1985, pp. 116–121.

[3] B. J. Kuipers and Y. T. Byun, "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *J. Robot. Auton. Syst.*, vol. 8, pp. 47–63, 1991.
[4] J. Matarić, "Integration of representation into goal-driven behavior-based robots," *IEEE Trans. Robot. Automat.*, vol. 8, pp. 304–312, Apr. 1992.
[5] S. Thrun, "Learning maps for indoor mobile robot navigation," *Artif. Intell.*, vol. 99, pp. 21–71, 1998.
[6] J. O'Keefe and L. Nadel, *The Hippocampus as a Cognitive Map*. Oxford, MA: Clarendon, 1978.
[7] N. Burgess, M. Recce, and J. O'Keefe, "A model of hippocampal function," *Neural Net.*, vol. 7, pp. 1065–1081, 1994.
[8] A. W. Moore and C. G. Atkeson, "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces," *Mach. Learn.*, vol. 21, pp. 199–233, 1995.
[9] S. Schaal, "Nonparametric regression for learning," in *Proc. Conf. Prerational Intell.—Adaptive Learning Behavior*, Bielefeld, Germany, 1994.
[10] J. del R. Millán, "Rapid, safe, and incremental learning of navigation strategies," *IEEE Trans. Syst., Man Cybern. A*, vol. 26, pp. 408–420, 1996.
[11] S. Schaal and C. G. Atkeson, "Assessing the quality of learned local models," in *Neural Information Processing Systems 6*, J. Cowan, G. Tesauro, and J. Alspector, Eds. San Mateo, CA: Morgan Kaufmann, 1994, pp. 160–167.
[12] D. A. Cohn, "Neural network exploration using optimal experiment design," *Neural Networks*, vol. 9, pp. 1071–1083, 1996.
[13] S. B. Thrun, "The role of exploration in learning control," in *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, D. A. White and D. A. Sofge, Eds. New York: Van Nostrand Reinhold, 1992.
[14] R. Chatila and J. P. Laumond, "Position referencing and consistent world modeling for mobile robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1985.
[15] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Trans. Robot. Automat.*, vol. 3, pp. 249–265, Apr. 1987.
[16] ———, *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*, Ph.D. dissertation, Dept. Elect. Comp. Eng., Carnegie Mellon Univ., Pittsburgh, PA, 1989.
[17] C. J. Taylor and D. J. Kriegman, "Exploration strategies for mobile robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, Los Alamitos, CA, 1993, vol. 2, pp. 248–253, IEEE Comp. Soc. Press.
[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.
[19] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1995.
[20] J. M. Renders, J. del R. Millán, and M. Becquet, "Non-geometrical parameters identification for robot kinematic calibration by use of neural network techniques," in *Proc. Eur. Robot. Intell. Syst. Conf.*, 1991.
[21] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C*. Cambridge, U.K.: Cambridge Univ. Press, 1992.
[22] S. Argamon, S. Kraus, and S. Sina, "Utility-based on-line exploration for repeated navigation in an embedded graph," *Artif. Intell.*, vol. 101, pp. 267–284, 1998.
[23] J. del R. Millán, and A. Arleo, "Neural network learning of variable grid-based maps for the autonomous navigation of robots," in *Proc. IEEE Int. Symp. Comp. Intell. Robot. Automat.*, 1997, pp. 40–45.
[24] J. Borenstein and L. Feng, "Correction of systematic odometry errors in mobile robots," in *Proc. Int. Conf. Intell. Robots Syst.*, 1995, pp. 569–574.
[25] K. Graves, W. Adams, and A. Schultz, "Continuous localization in changing environments," in *Proc. IEEE Int. Symp. Comp. Intell. Robot. Automat.*, 1997, pp. 28–30.
[26] B. Yamauchi and P. Langley, "Place recognition in dynamic environments," *J. Robot. Syst.*, vol. 14, pp. 107–120, 1997.

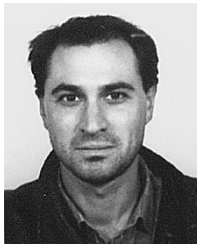


Angelo Arleo received the M.S. degree in computer science from the University of Mathematical Science, Milan, Italy, in 1996 and is currently pursuing the Ph.D. degree at the Swiss Federal Institute of Technology Lausanne (EPFL), Lausanne, Switzerland.

He is a Researcher at the EPFL. In 1995, he worked with the Robot Learning Group, Joint Research Centre, European Commission, Italy. His research topics concern spatial modeling and navigation in neuro-mimetic systems, adaptive robotics, and artificial neural networks.

José del R. Millán received the Ph.D. degree in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1992.

He is a Research Scientist with the Joint Research Centre, European Commission. He was an Assistant Professor for three years with the Universite Politècnica de Catalunya, Barcelona, for three years. His work is mainly focused on the design of adaptive agents, in particular autonomous robots and wearable biosignal-based human-computer interfaces.



Dario Floreano received the M.S. degree in neural computation from the University of Stirling, U.K., and the Ph.D. degree in cognitive science, from the University of Trieste, Trieste, Italy, in 1992 and 1995, respectively.

He is a Senior Researcher with the Swiss Federal Institute of Technology Lausanne (EPFL), Lausanne, Switzerland, with interests in autonomous robots, evolutionary systems, and artificial life. Since 1993, he has published about 50 technical peer-reviewed papers and books, organized two International Conferences (ECAL99 and SAB2000), and joined the scientific committee of more than 20 conferences and journals.