MAKING ROBOTS SMART

Behavioral Learning Combines Sensing and Action

EDITED BY

Attilio Giordana
Università di Torino
Torino, Italy

Michael Kaiser
ABB Corporate Research Ltd.
Baden-Daettwil, Switzerland

Volker Klingspor
Universität Dortmund
Dortmund, Germany

Hendrik Van Brussel
Katholieke Universiteit Leuven
Leuven, Belgium

KLUWER ACADEMIC PUBLISHERS
Boston/London/Dordrecht

CONTENTS

CONTRIBUTORS

Marco Accame
University of Genoa
Department of Biophysical and Electronic Engineering (DIBE)
16145 Genova, Italy

Manuel Barata
Universidade Nova de Lisboa
Departamento de Engenharia Electrotécnica
2825 Monte da Caparica, Portugal

Cristina Baroglio
Università di Torino
Dipartimento di Informatica
10149 Torino, Italy

Luis Basañez
Universitat Politècnica de Catalunya
Institut de Cibernètica
08028 Barcelona, Spain

Luís M. Camarinha-Matos
Universidade Nova de Lisboa
Departamento de Engenharia Electrotécnica
2825 Monte da Caparica, Portugal

Rüdiger Dillmann
Universität Karlsruhe
Institute for Real-Time Computer Systems & Robotics
76128 Karlsruhe, Germany

Holger Friedrich
Universität Karlsruhe
Institute for Real-Time Computer Systems & Robotics
76128 Karlsruhe, Germany

Attilio Giordana
Università di Torino
Dipartimento di Informatica
10149 Torino, Italy

Michael Kaiser
ABB Corporate Research Ltd.
Information Technology Dept.
5405 Baden-Daettwil, Switzerland

Volker Klingspor
Universität Dortmund
Lehrstuhl Informatik VIII
44221 Dortmund, Germany

Jose del R. Millan
Joint Research Centre of the European Commission
Institute for Systems, Informatics and Safety
21020 Ispra, Italy

Katharina Morik
Universität Dortmund
Lehrstuhl Informatik VIII
44221 Dortmund, Germany

Marnix Nuttin
Katholieke Universiteit Leuven
Department of Mechanical Engineering, Division PMA
3001 Leuven, Belgium

Roberto Piola
Università di Torino
Dipartimento di Informatica
10149 Torino, Italy

Thomas W. Rauber
Universidade Federal do Espírito Santo
Departamento de Informática
29065-900 Vitória - ES, Brazil,

Anke Rieger
Universität Dortmund
Lehrstuhl Informatik VIII
44221 Dortmund, Germany

Jan Rosell
Universitat Politècnica de Catalunya
Institut de Cibernètica
08028 Barcelona, Spain

Luís Seabra Lopes
Universidade de Aveiro
Departamento de Electrónica e Telecomunicações
3810 Aveiro, Portugal

Raúl Suárez
Universitat Politècnica de Catalunya
Institut de Cibernètica
08028 Barcelona, Spain

Carme Torras
Universitat Politècnica de Catalunya
Institut de Cibernètica
08028 Barcelona, Spain

Hendrik Van Brussel
Katholieke Universiteit Leuven
Department of Mechanical Engineering, Division PMA
3001 Leuven, Belgium

Stephanie Wessel
Universität Hamburg
Labor für Künstliche Intelligenz
Vogt-Kölln-Str. 30
22527 Hamburg, Germany

7

LEARNING SENSOR-BASED NAVIGATION

J. del R. Millán and C. Torras*

Joint Research Centre of the European Commission
**Universitat Politècnica de Catalunya*

ABSTRACT

A mobile robot that uses reinforcement learning to acquire reactive navigation skills is presented. The basic skills needed for reaching a goal while avoiding obstacles are encoded as sensation-action associations in a modular Neural Net. The robot has no a priori knowledge of either the environment or the effect of its actions, and learns on-line using only raw sensory data collected as it moves. The experimental results show that a few trials suffice for the robot to navigate efficiently in a real environment of moderate complexity.

1 INTRODUCTION

Mobile robots must be able to navigate autonomously if they are to perform useful tasks in unconstrained environments. Moreover, since these environments are usually unknown when faced for the first time, robots need to be endowed with learning capabilities. This chapter deals with the learning task of acquiring a set of safe and efficient elementary operations for navigation. By elementary we mean that the learning task is aimed at closing the sensing-action loop at the lowest control level of the autonomous vehicle. In other words, the robot is to learn the suitable *reactive motor skills* using only *raw sensory data*. In particular, these skills correspond to *goal-oriented obstacle-avoidance reactions* since the robot has to navigate efficiently. Some crucial aspects of this learning task are the following. First, the robot must always make decisions in *real time*. A robot operating in a hazardous environment cannot afford to stop long times to select the best course of action and/or update its current knowledge about the task. Second, the robot's knowledge has to be *grounded*. In fact, since the robot does not have any a priori knowledge of the environment, it can only work upon information that is extracted from its sensors. Third, the robot's controller (and all related modules) has to deal with *noisy sensory data*.

One way in which a robot can autonomously learn reactive motor skills is to first acquire a model of the effects of its actions. The robot can then employ this model to make a plan to achive the goal. Examples of this approach are the works of Atkeson [Atkeson, 1991], Moore [Moore, 1991] and Thrun et al. [Thrun *et al.*, 1991]. Nevertheless, building and maintaining good enough models is not only computationally expensive, but also prone to errors if built from noisy sensory data.

A different approach is to learn the suitable reactions directly from the interactions with the environment. The robot simply tries different actions for every situation it finds when experiencing the environment and selects the most useful ones as measured by a *reinforcement* or performance feedback signal. Indeed, *reinforcement learning (RL)* is thought to be an appropriate paradigm to acquire control policies for autonomous robots that work in initially unknown environments. Many of the tasks faced by such robots have absorbing goal states, and so their aim is to learn to perform those actions that maximize the cumulative reinforcement in the long-term (i.e., from the moment an action is taken until the goal is reached). These robots must learn a policy that maximizes $V(t) = \sum_{k=0}^{T} z(t+k)$, where $z(\tau)$ is the reward obtained at time τ and $V(t)$ is the total future reinforcement from time t until the goal is reached at time T.

Most RL systems learn *value functions* by means of *temporal difference (TD) methods* [Sutton, 1988]. A value function estimates V for situations or for situation-action pairs. *Actor-Critic* architectures [Barto *et al.*, 1983] and *Q-learning* [Watkins, 1989] are two of these RL systems. Recent works have proven the asymptotic convergence of TD methods (e.g., [Dayan and Sejnowski, 1994]). Unfortunately, these proofs rely on several assumptions that hardly apply to robots facing real-world tasks. In particular, most of them require to codify the TD's estimates into tabular representations (which implies discrete situation and action states)[a] and to try out every action for every situation

[a] Some results apply to non-tabular representations [Tsitsiklis and Van Roy, 1996], but rely on the use of appropriate features that are typically hand-coded. Interestingly, the authors discuss how their feature-based method may be combined with local networks similar to ours. On the contrary, theoretical results (e.g., [Gordon, 1995]) have shown that TD methods may fail to converge if global function approximators, such as feedforward NN, are used.

an infinite number of times (which is always unfeasible, and can only be approximated if both spaces are small). In many real-world tasks, however, the situation and action spaces are continuous and robots cannot afford long, risky learning trials. Thus, practical learning robots require compact representations such as NN to generalize experience between similar situations and actions, and to limit their experience to relevant parts of the problem only. But, what can we expect from reinforcement-based agents that do not satisfy the theoretical requirements? Our experimental results with the autonomous mobile robot TESEO demonstrate empirically that, despite the lack of convergence results, reinforcement-based robots can rapidly solve real-world tasks. See also chapter 5 for other examples of successful reinforcement-based robots.

Key components of our learning architecture are the use of *local networks*[b] and the incorporation of *bias* into the network. The local networks we advocate for allow the robot to learn incrementally so as to adapt to new or changing environments without degrading its performance in previous situations. See 3 for a more detailed explanation of the advantages of local networks and incremental learning. We use built-in reflexes (domain knowledge) as bias. Two are the benefits of bias. First, it accelerates the learning process since it focusses the search process on promising parts of the action space immediately. Second, it makes the robot operational from the very beginning and increments the safety of the learning process.

Categorization

Learning task: The robot has to learn autonomously to reach a goal while avoiding obstacles in an unknown environment. This amounts to learning suitable *reactive motor skills* on the basis of only *raw sensory data* and some *built-in reflexes*. In particular, these skills correspond to *goal-oriented obstacle-avoidance reactions*, which are computed by means of an *unstructured numerical function* that maps perceived situations onto suitable actions.

Training data generation: As the robot moves in a *real environment*, it will face *thousands* of sensory situations for which to perform actions. Each such situation constitutes a training instance, which has associated a data pair consisting of the set of readings from the robot sensors and the set of motor commands executed. In this way, *non-labeled training data* of an *homogeneous nature* are generated. Moreover, *training data are noisy*

[b]Indeed, Sutton [Sutton, 1996] reports that TD converges if the function approximator is a sparse coarse coding network instead of a global one.

and lack any structure, since the robot works upon raw sensory data and performs basic actions. An additional source of noise is the odometry system.

Choice of learning technique: The mobile robot must not only perform actions autonomously, but also *learn autonomously* using the *environment as its teacher*. Moreover, the robot has *no a priori knowledge* about either the environment or the effects of its actions, and must only use information about what it can actually perceive. Finally, the robot has to learn *incrementally* and *on-line*. These requirements point to neural reinforcement learning as the most suitable approach.

Evaluation of learning result: The performance of the learning robot is evaluated on-line in terms of the quality of the trajectory; that is, length and clearance to the obstacles.

Adaptation of learning result: Since the robot is learning autonomously, incrementally and on-line, it adapts continuously its behavior.

Constraints: Learning must take place as the robot moves. This imposes a *strong time constraint* if the robot is to act in real time.

2 ROBOT TESTBED

TESEO is a commercial *Nomad 200* mobile robot (figure 1). It has three independent motors: the first moves the three wheels of the robot together, the second steers the wheels together, and the third rotates the turret of the robot. The robot is equipped with 16 infrared sensors, 16 sonar sensors, and 20 contact sensors. The infrared and sonar sensors are evenly placed around the perimeter of the turret and the tactile sensors cover all the perimeter of the robot below the turret. Moreover, the robot has a dead-reckoning system that keeps track of the robot's position and orientation.

TESEO is controlled by a NN that maps the currently perceived situation into the next action. Its task is to reach a given goal location specified in cartesian coordinates. A situation is made of sensory information coming from physical as well as virtual sensors. The action determines the next direction of travel. TESEO receives a reinforcement signal after performing every action. This assumption is hard to satisfy in many reinforcement learning tasks, but not in those where the robot can estimate the state of achievement at any moment. In our case, knowing the goal location permits rewarding actions approaching

Figure 1 The mobile robot TESEO, a commercial Nomad 200.

it while avoiding obstacles. It is important to note, however, that TESEO does not seek to optimize each immediate reinforcement signal, but to optimize the total amount of reinforcement obtained along the path to the goal.

While moving, TESEO uses a low-level asynchronous emergency routine to prevent collisions. The robot stops and retracts whenever its range sensors detect an obstacle in front of it which is closer than a safety distance or its tactile sensors detect a collision. In this case, the learning algorithm penalizes the action that makes the emergency routine intervene.

The input to the NN consists of a vector of 40 real numbers in the interval $[0, 1]$. The first 32 components correspond to the infrared and sonar sensor readings. In this case, a value close to zero means that the corresponding sensor is detecting a very near obstacle. The remaining 8 components correspond to a coarse codification of an inverse exponential function of the distance from the current robot location to the goal, as computed by a virtual sensor based on the dead-reckoning system. The incorporation of information about the goal into the input pattern allows TESEO to discriminate between similar sensory situations that require different suitable actions.

The output of the NN consists of a single component that controls directly the steering motor and indirectly the turret rotation motor. This component is a real number in the interval $[-180, 180]$ and determines the direction of travel with respect to the vector connecting the goal and the current robot location. Once the robot has steered the commanded degrees, it translates a fixed distance (25 cm) and, at the same time, it rotates its turret in order to maintain the front infrared and sonar sensors oriented toward the goal. It is worth noting that a codification of both the physical sensor readings and the motor command relative to the goal location enhances TESEO's generalization capabilities. This relative codification is not only a "technical trick", possible in a robot like the Nomad 200, where the turret orientation can be different from the robot's direction of travel. There exists also experimental evidence that animals learn goal-oriented actions rather than specific motor commands [Roitblat, 1994].

The reinforcement signal z is a real number in the interval $[-3, 0]$ which measures the *cost* of doing a particular action in a given situation. The cost of an action is directly derived from the task definition, which is to reach the goal along trajectories that are sufficiently short and, at the same time, have a wide clearance to the obstacles. The reward is -3 if TESEO collides or detects an obstacle closer than a safety distance, and it is 0 if TESEO moves straight to the goal and does not detect near obstacles.

3 THE LEARNING APPROACH

TESEO's learning architecture is an improved Actor-Critic system. The Critic uses TD methods to estimate the total future reinforcement that will be obtained if the robot performs the best currently known actions that take it from its current location to the goal. The Actor uses this estimation to update the situation-action mapping, codified into the NN, using the *associative search (AS)* learning rule [Barto *et al.*, 1983; Williams, 1992]. The improvements to the standard Actor-Critic architecture allows TESEO to overcome three limitations of basic reinforcement learning: slow convergence, unsafe process, and lack of incremental improvement. The next three subsections present three aspects of TESEO's architecture that address these limitations.

3.1 Learning from basic reflexes

The main reason for the slow convergence of basic reinforcement learning is that every time the learner faces a new situation it takes random actions. A way of overcoming this problem is to use domain knowledge to determine quickly where to search the suitable action to handle each situation.

Along this line, instead of learning from scratch, TESEO searches within a fixed set of *basic reflexes* every time its neural controller fails to generalize correctly its previous experience to the current situation. The neural controller associates the selected reflex with the perceived situation in one step. The sensory situation is represented by a new unit of the NN and the selected reflex is codified into the weights of the controller. This new reaction rule is tuned subsequently through reinforcement learning. As explained later in the chapter, incorrect generalizations occur when none of the learned reaction rules is applicable to the perceived situation or when the actual reinforcement obtained after taking an action is much worse than expected. In this way, the NN gets control (and thus suppresses the activation of the basic reflexes) more often as the robot explores the environment.

Basic reflexes correspond to previous knowledge about the task and are codified as simple reactive behaviors [Brooks, 1986]. As the actions computed by the neural controller, the reflexes determine the next direction of travel which is followed for a fixed distance while the robot rotates its turret. Each basic reflex selects one of the 16 directions corresponding to the current orientations of the infrared and sonar sensors. We have chosen this fixed set of directions because they are the most informative for the robot in terms of obstacle detection. The basic reflexes are prewired into a subsumption-like architecture with three reactive behaviors. These behaviors are, in order of priority, collision avoidance, move to goal and track object boundary (figure 2). Every time that the basic reflexes are invoked, the behavior with the highest priority that is applicable to the current situation is activated. The behaviors collision avoidance and track object boundary are inspired in those of [Mataric, 1992] and their combination makes TESEO follow safely the boundaries of objects in the environment.

It is worth noting that, except in simple cases, these reflexes alone do not generate efficient trajectories; they just provide acceptable starting points for the reinforcement learning algorithm to search appropriate actions. TESEO does not learn to coordinate preprogrammed reflexes; rather it learns new, efficient reactions from basic reflexes.

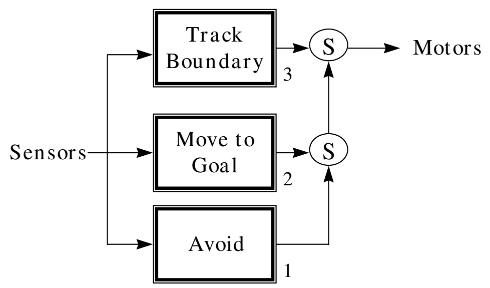

Figure 2 Codification of the basic reflexes into a subsumption-like architecture with 3 reactive behaviors. Applicable behaviors with higher priorities suppress other behaviors with lower priorities.

Integrating learning and reaction in this way allows TESEO to focus on promising parts of the action space immediately, which accelerates learning. In addition, learning on top of basic reflexes makes TESEO operational from the very beginning and increments the safety of the learning process.

3.2 Modularity

TESEO's controller is a *modular network*. Each module maps similar sensory inputs into similar actions and, in addition, they have similar long-term consequences. A *resource-allocating procedure* adds new units to the appropriate module only when necessary to improve TESEO's performance. This procedure is based mainly on the work of Alpaydin [Alpaydin, 1991]. The main difference is that Alpaydin's approach is intended for supervised learning tasks, while ours is for reinforcement learning tasks.

Modularity guarantees that improvements in the response to a given situation will not negatively alter other unrelated reactions, thus achiving incremental learning.

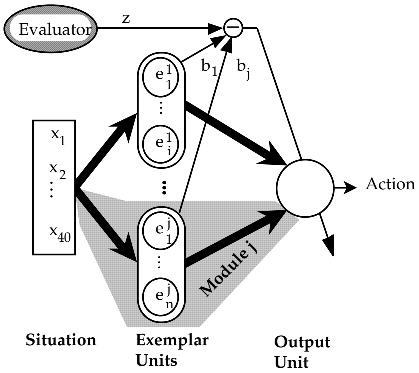

Figure 3 Controller architecture.

3.3 Exploration

TESEO explores the action space by concentrating the search around the best actions currently known. The width of the search is determined by a *counter-based scheme* associated to the modules (see section 4.2). This exploration technique allows TESEO to avoid experiencing irrelevant actions and to minimize the risk of collisions.

4 CONTROLLER ARCHITECTURE

The neural controller is a modular two-layer network (figure 3). The first layer consists of units with *overlapping localized receptive fields*, which we call *exemplars*. The second layer is made of one single stochastic linear unit, the output unit. Big arrows in figure 3 represent a full connectivity between the components of the input vector and the exemplars and between the exemplars and the output unit.

4.1 Modules and exemplars

Since modules are created dynamically as TESEO explores its environment, different modules can have different number of exemplars. For instance, module 1 in figure 3 has i exemplars while module j has n exemplars. Every module j keeps track of four adaptive values that are updated during learning. First, the expected total future reinforcement, b_j, that the robot will receive if it uses this module for computing the next action. Second, the width of the receptive fields of all the exemplars in this module, d_j. Third, a counter that records how many times this module has been used without improving the robot's performance, c_j. Fourth, the prototypical action the robot should normally take whenever the perceived situation is classified into this module, pa_j.

Every exemplar e_k^j is a tuple $(\mathbf{v}_k^j, d_j)$ which represents a sphere in the input space. This sphere is centered on the point $\mathbf{v}_k^j \in [0,1]^{40}$, and has a radius (or width of the receptive field) d_j. The activation level a_k^j of the exemplar e_k^j is a real value in the interval $[0,1]$ that measures how well e_k^j matches the current perceived situation $\mathbf{x}$. The activation level is 0 if the perceived situation is outside the receptive field of the exemplar, and 1 if the situation corresponds to the point where the exemplar is centered. As mentioned above, exemplars have overlapping receptive fields. Hence a given situation can activate several exemplars simultaneously, each at a different degree.

Actions are computed after a *competitive process* among the existing modules: only the module that best classifies the perceived situation propagates the activation levels of its exemplars to the output unit. The winning module, if any, is the one having the highest active exemplar. In the case that no exemplar "matches" the perceived situation —i.e., if the input does not fall in the receptive field of any exemplar—, then a basic reflex is triggered and the current situation becomes a new exemplar. Section 5.2 provides more details about the resource-allocating procedure.

After reacting, the *evaluator* computes the reinforcement signal, z. Then, if the action was computed through the module j, the difference between z and b_j is used for learning. Only the weights of the links associated to the winning module j are modified.

Thus, in order to improve its reactions to a particular kind of situations, TESEO needs just to adapt the module that covers that region of the input space. The adaptation of a module is done either by adding more exemplars or by tuning its weights.

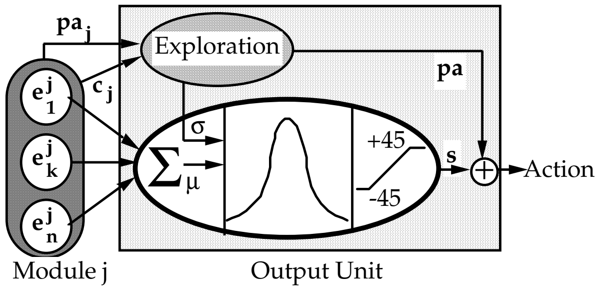

Figure 4 The output unit. For the sake of simplicity, this figure only shows the information sent by the winning module.

4.2 Output unit

As illustrated in figure 4, the output of the neural controller is a prototypical action pa plus a certain variation s.

As mentioned in section 3.3, the exploration needed for reinforcement learning is carried out only around the best actions currently known. Since each action is the sum of two components, pa and s, the exploration mechanism works on each of them separately.

On the one hand, and assuming j is the winning module, if c_j is not divisible by a constant k_{exp}, then pa_j is chosen. Otherwise, pa is taken to be the prototypical action associated to the module m activated by the current situation that has the best expected total future reinforcement, b_m. The value of k_{exp} was empirically chosen to be 3 in the experiments.

On the other hand, the deviation s from pa is computed through a stochastic process in such a way that TESEO will only explore actions between pa and its four neighboring prototypical actions (two to the left and two to the right). The computation of s is done in three steps.

The first step is to determine the value of the stochastic process' parameters. The mean μ is a weighted sum of the activation levels of the exemplars $e_1^j, \ldots, e_n^j$

of the winning module:

$$\mu = \sum_{k=1}^{n} w_k^j a_k^j, \tag{6.1}$$

where w_k^j is the weight associated to the link between e_k^j and the output unit, and a_k^j is the activation level of e_k^j. The variance σ is proportional to c_j. This follows from the idea that the most often the module j is used without improving TESEO's performance, the higher σ must be.

In the second step, the unit calculates its *activation level* l which is a normally distributed random variable:

$$l = N(\mu, \sigma). \tag{6.2}$$

In the third step, the unit computes s:

$$s = \begin{cases} 45, & \text{if } l > 45, \\ -45, & \text{if } l < -45, \\ l, & \text{otherwise.} \end{cases} \tag{6.3}$$

5 LEARNING MECHANISMS

There are four basic forms of learning in the proposed architecture. The first is related to the update of the b_j. The second kind of learning regards the topology of the network. The third type of learning consists on tuning the position of the exemplars. Finally, the fourth concerns weight modification.

5.1 Improving reinforcement estimates

Let us remind that b_j is an estimate of the total future reinforcement TESEO will obtain if it performs the best currently known actions that take it from its current location (whose associated observed situation is classified into the j^{th} module) to the goal. Consequently, the value b_j of the module j should, after learning, be equal to the sum of the cost z of reaching the best next module i plus the value b_i. Since z takes negative values, minimizing future cost corresponds to maximizing future reinforcement:

$$b_j = \max_{y \in Actions}(z) + b_i. \tag{6.4}$$

During learning, however, this equality does not hold for the value b_j of every module j because the optimal action is not always taken and, even if it is, the value b_i of the next module i has not yet converged.

In order to iteratively update the values of b_j, so that finally (6.4) holds for all of them, we have used the simplest TD method, i.e. TD(0) [Sutton, 1988]. If the situation perceived at time t is classified into the module j and, after performing the computed action, the next situation belongs to the module i and the reinforcement signal is $z(t+1)$, then:

$$b_j(t+1) = b_j(t) + \eta * \left[z(t+1) + b_i(t) - b_j(t) \right]. \qquad (6.5)$$

η controls the intensity of the modification, taking the value 0.75 when TESEO behaves better than expected, and 0.075 in the opposite case. The rationale for modifying less intensively b_j when $z(t+1) + b_i(t) - b_j(t) < 0$ is that this error is probably due to the selection of an action different from the best currently known for the module j.

The problem is for TESEO to figure out $b_i(t)$ when no module is activated by the next situation. In this case, b_i is estimated on the basis of the distance from the next location to the goal and the distance to the perceived obstacles in between the robot and the goal.

5.2 Network growth

The second learning mechanism makes the NN grow as a function of the inputs received. Initially, there exist neither exemplars nor, consequently, modules and the resource-allocating procedure creates them as they are needed.

As mentioned in section 4.1, if no exemplar "matches" the perceived situation, then a basic reflex is triggered and the current situation becomes a new exemplar. The weight of the link from this exemplar to the output unit is initially set to zero and evolves subsequently through reinforcement learning.

The new exemplar is added to one of the existing modules if its receptive field overlaps the receptive fields of the module's exemplars and the selected reflex is the same as the module's prototypical action. The first condition assures that every module will cover a connected input subspace.

If any of the two conditions above is not satisfied, then a new module consisting of this exemplar and its associated connections is created. Concerning the four parameters associated to this new module m, they are initially set to the following values: d_m equals k_d, c_m equals 0, pa_m is the selected reflex, and b_m is estimated as when no module is activated by the perceived situation (see the preceding section). The value of k_d is 0.4, which corresponds to 1% of the whole input space.

5.3 Tuning exemplars

The third learning mechanism moves the position of the exemplars $e_1^j, \ldots, e_n^j$ of the winning module j in order to better cover the input subspace dominated by that module. That is, the coordinates of the k^{th} exemplar, $\mathbf{v}_k^j$, are updated proportionally to how well they match the perceived situation $\mathbf{x}$:

$$\mathbf{v}_k^j(t+1) = \mathbf{v}_k^j(t) + \epsilon * a_k^j(t) * \left[\mathbf{x}(t) - \mathbf{v}_k^j(t)\right], \tag{6.6}$$

where ϵ is the learning rate. In the experiments reported below, the value of ϵ is 0.1.

5.4 Weight update

The classical *associative search (AS)* [Barto *et al.*, 1983; Williams, 1992] rule is applied to the weights of the connections from the exemplars $e_1^j, \ldots, e_n^j$ to the output unit:

$$w_k^j(t+1) = w_k^j(t) + \beta * \left[z(t+1) + b_i(t) - b_j(t)\right] * \phi_k^j(t), \tag{6.7}$$

where β is the learning rate, and ϕ_k^j is the eligibility factor. The intensity of the weight modifications depends on the relative merit of the action, which is just the error provided by the TD method.

The eligibility factor of a given weight measures how influential that weight was in choosing the action. In our experiments, ϕ_k^j is computed in such a manner that the learning rule corresponds to a *gradient ascent* mechanism on the expected reinforcement [Williams, 1992]:

$$\phi_k^j(t) = \frac{\partial lnN}{\partial w_k^j}(t) = a_k^j(t)\frac{l(t) - \mu(t)}{\sigma^2(t)}, \tag{6.8}$$

where N is the normal distribution function in (6.2). The weights w_k^j are modified more intensely in case of *reward* —i.e., when TESEO behaves better than expected— than in case of *penalty*. These two values of β are 0.2 and 0.02, respectively. The aim here is that TESEO maintains the best situation-action rules known so far, while exploring other reaction rules.

6 LEARNING OPPORTUNITIES

Let us present now the four occasions on which learning takes place. The first arises during the classification phase, the next two happen after reacting, and the last one takes place when reaching the goal.

6.1 Unexperienced situation

If the perceived situation is not classified into one of the existing modules, then the basic reflexes get control of the robot, and the resource-allocating procedure creates a new exemplar which is added either to one of the existing modules or to a new module.

6.2 Performing within expectations

If the perceived situation is classified into the module j and $z(t+1) + b_i(t) - b_j(t) \geq k_z$, where k_z is a negative constant, then (i) the exemplars of that module are tuned to make them closer to the situation, (ii) the weights associated to the connections between the exemplars and the output unit are modified using the AS reinforcement learning rule, (iii) b_j is updated through TD(0), and (iv) d_j, c_j, and pa_j are adapted.

The adaptive parameters are updated differently in case of reward than in case of penalty. In case of reward, d_j is increased by 0.1, c_j is initialized to 0, and if the output of the neural controller, $\alpha = pa + s$, is closer to a prototypical action $\xi(t)$ other than $pa_j(t)$, then $pa_j(t+1) = \xi(t)$. In case of penalty, c_j is increased by 1 and d_j is decreased by 0.1 if it is still greater than the threshold $k_d/2$, where k_d is the initial value of d_j.

6.3 Performing rather badly

If the perceived situation is classified into the module j and $z(t+1) + b_i(t) - b_j(t) < k_z$, then the topology of the network is slightly altered and d_j is decreased by 0.1 if it is still greater than the threshold $k_d/2$.

If the total future reinforcement computed after reacting, $z + b_i$, is considerably worse than the expected one, b_j, this means that the situation was incorrectly classified and needs to be classified into a different module.

The resource-allocating procedure creates a new exemplar, e_u, that has the same coordinates as the perceived situation, but it does not add it to any module. The next time this situation (or a similar one) will be faced, e_u will be the closest exemplar. Consequently, no module will classify the situation and the basic reflexes will get control of the robot. Then, the resource-allocating procedure will add e_u either to one of the existing modules or to a new module as described in section 5.2.

This means that TESEO classifies situations, in a first step, based on the similarity of their input representations. Then, it also incorporates task-specific information for classifying based on the similarity of reinforcements received.

6.4 Reaching the goal

Finally, whenever the goal is reached, the values b_j and pa_j of every winning module j along the path to the goal are also updated in reverse chronological order. To do that, TESEO stores the triples $\langle j(t), \alpha(t), z(t+1) \rangle$ along the current path, where $j(t)$ indicates the winning module at time t. Then, after reaching the goal at time $n+1$ and updating the last value $b_{j(n)}$, the value $b_{j(n-1)}$ is updated, and so on until $b_{j(1)}$. This technique only accelerates the convergence of the value b_j of every module j, but does not change its steady value [Mahadevan and Connell, 1992].

Regarding the update of the value $pa_{j(t)}$, it is done before updating the value $b_{j(t)}$. That is, if $z(t+1) + b_{j(t+1)} - b_{j(t)} \geq k_z$ and $\alpha(t)$ is closer to a prototypical action $\xi(t)$ other than $pa_j(t)$, then $pa_j(t+1) = \xi(t)$.

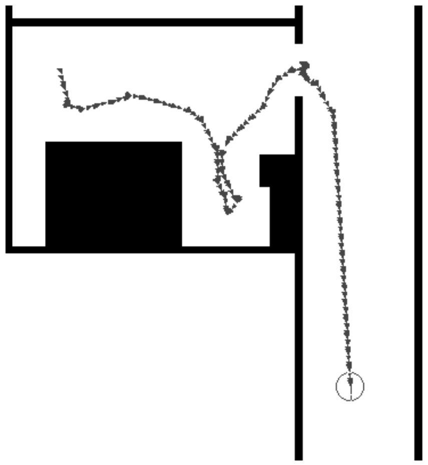

Figure 5 The environment and first trajectory generated for a starting location within the office. Note that TESEO has some problems in going through the doorway.

7 EXPERIMENTAL RESULTS

TESEO's performance has been tested on a corridor with offices at both sides. The task is to generate a short but safe trajectory from inside an office (of size 4.5×3.5 meters) to a point at the end of a corridor (1.8 meters wide). The office is cluttered with furnitures and, occasionally, static boxes. TESEO achieves the target location every time and it never gets lost or trapped inside concave obstacles. The first time it tries to reach the goal it relies almost all the time on the basic reflexes, which make TESEO follow walls and move around obstacles. As illustrated in figure 5, in the first trial, TESEO enters into a dead-end section of the office (but it does not get trapped into it) and even it collides against the door frame because its sensors were not able to detect it. Collisions happened because the frame of the door is relatively thin and the incident angles of the rays drawn from the sonar and infrared sensors were too large resulting in specular reflections.

TESEO learns to follow a smooth trajectory to the goal very quickly, in just ten trials (figure 6). The resulting neural controller consists of 48 modules and 89 exemplars.

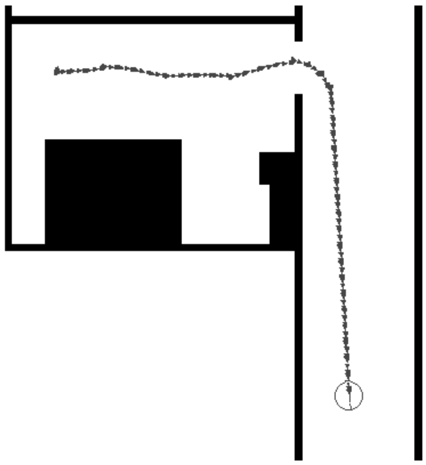

Figure 6 Trajectory generated after travelling ten times to the goal.

This experiment was run fifteen times, obtaining similar results. figures 7, 8 and 9 show the learning curves for one of the runs and table 1 reports the mean and standard deviation of the final performances over the fifteen runs. Note that the performance improves from the first trajectory to the seventh. In the eighth trajectory, TESEO explores new actions that turn out to be inappropriate. Then, TESEO recovers its preceding level of performance. Note also that after the tenth trajectory TESEO's performance stabilizes. Figure 9 illustrates the number of exemplars and modules of the neural controller after the generation of each trajectory.

	Mean	**Standard Deviation**
Total Reinforcement	-6.52	0.2108
Steps	42.8	0.8718
Exemplars	88.6	2.0010
Modules	47.6	1.9596

Table 1 Mean and standard deviation of the final performances over ten runs.

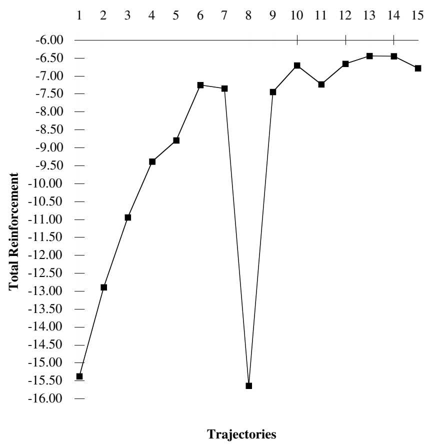

Figure 7 Total reinforcement obtained along each trajectory.

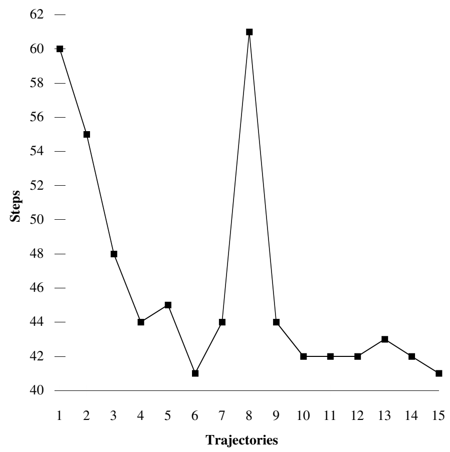

Figure 8 Number of steps taken along each trajectory.

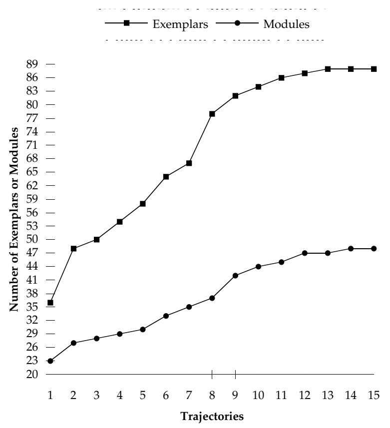

Figure 9 Number of exemplars and modules of the neural controller after the generation of each trajectory.

The resulting trajectories are quite smooth even if the basic reflexes have been programmed in a discretized way.

Figure 10 illustrates instances of the reaction rules learned. For every location considered (little circles) the move to be taken is depicted. Figure 10 shows that TESEO generates solution paths from any starting location inside the room. This indicates that TESEO exhibits good generalization abilities, since it can handle many more situations than those previously perceived.

Once TESEO has learned efficient navigation strategies, occasional static obstacles are put in the way to the goal. As illustrated in figure 11, TESEO moves around the obstacles and then it returns to the original, efficient trajectory. In other experiments, the goal location is changed. TESEO learns also to navigate to this new goal in a few trials and it is still able to reach the first goal as efficiently as before.

However, a note of caution should be stated when talking about generalization capabilities since, because of its modular architecture and the use of exemplars with localized receptive fields, TESEO may not be able to navigate in radically new environments.

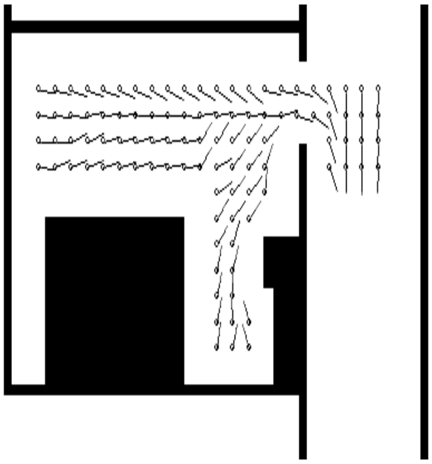

Figure 10 Generalization abilities: Situation-action rules applied for a sample of locations within the office and the first part of the corridor.

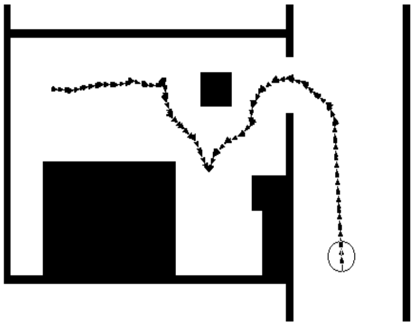

Figure 11 TESEO's behavior when facing an unexpected obstacle.

To finish this section, we wish to mention that, given the robot testbed used in the reported experiments, we have only used static goals. But Millán [Millán, 1995; Millán, 1997] presents results with simulated and real robots that show how our approach deals with moving targets.

8 CONCLUSIONS

We have described a reinforcement learning architecture that allows an autonomous mobile robot to acquire efficient navigation strategies in a few trials. Besides rapid learning, the architecture has three further appealing features. First, the robot improves its performance incrementally as it interacts with an initially unknown environment, and it ends up learning to avoid collisions even in those situations in which its sensors cannot detect the obstacles. This is a definite advantage over non-learning reactive robots. Second, since it learns from basic reflexes, the robot is operational from the very beginning and the learning process is safe. Third, the robot exhibits high tolerance to noisy sensory data and reasonable generalization abilities. All these features make this learning robot's architecture very well suited to real-world applications. For more details about the learning architecture, see [Millán, 1996].

Our approach suffers, however, from some limitations. The first limitation is that the reactive rules learned in a given environment do not generalize across new environments. This is a consequence of both the modular architecture and the use of exemplars with localized receptive fields: each module can only generalize locally. But the robot can rapidly adapt to the new environments while maintaining the same level of performance in the original one. Indeed, there exists a trade-off between generalization, on the one hand, and rapid, safe and incremental learning, on the other hand. The experimental results reported in this chapter illustrate one aspect of this trade-off, while Millán and Torras [Millán and Torras, 1992] illustrate the opposite aspect. They have proposed a neural controller for a simulated mobile robot that shows good generalization capabilities when facing new environments, but it requires much more time to learn, does not improve the performance incrementally, and applies its current knowledge to any new situation without any caution.

Another limitation is the number of parameters of the architecture that need to be manually tuned. Millán [Millán, 1997] describes a conceptually simpler architecture.

The main limitation is the strong dependence on a reliable *odometry system* that keeps track of the robot's relative position with respect to the goal. In the reported experiments, odometry is totally based on dead-reckoning and thus the goal has to be fixed and specified in cartesian coordinates. In all the experiments we have carried out so far, dead-reckoning has proven sufficient to reach the goal. As long as its estimation of the position of the robot does not differ greatly from the actual one, the neural controller is still able to produce correct actions. But, dead-reckoning will probably be insufficient in large environments. Millán [Millán, 1997] describes the incorporation of photosensitive sensors that allow the robot to recognize the goal and track it while moving. Furthermore, Millán and Arleo [Millán and Arelo, 1997] have developed a complementary NN technique to learn a map of an unknown structured environment incrementally and on-line. It creates a variable resolution partitioning of the environment, from which a topological map is learned on-the-fly. It is worth noting that, once the environment is 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.