Efficient Reinforcement Learning of Navigation Strategies in an Autonomous Robot

José del R. Millán
Institute for Systems Engineering and Informatics
European Commission. Joint Research Centre
TP 361. 21020 Ispra (VA). ITALY
e-mail: jose.millan@cen.jrc.it

Carme Torras
Institut de Cibernètica (CSIC-UPC)
Diagonal, 647. 08028 Barcelona. SPAIN
e-mail: torras@ic.upc.es

Abstract—**In this paper we propose a reinforcement learning architecture that allows an autonomous robot to acquire efficient navigation strategies in a few trials. Besides fast learning, the architecture has 3 further appealing features. (1) Since it learns from built-in reflexes, the robot is operational from the very beginning. (2) The robot improves its performance incrementally as it interacts with an initially unknown environment, and it ends up learning to avoid collisions even if its sensors cannot detect the obstacles. This is a definite advantage over non-learning reactive robots. (3) The robot exhibits high tolerance to noisy sensory data and good generalization abilities. All these features make this learning robot's architecture very well suited to real-world applications. We report experimental results obtained with a real mobile robot in an indoor environment that demonstrate the feasibility of this approach.**

I. Introduction

Efficient navigation is critical for autonomous robots operating in hostile environments, which are usually unknown the first time robots face them. This paper deals with the problem of controlling an autonomous mobile robot so that it reaches efficiently a goal location in an unknown indoor environment. Instances of the problem where the goal is not inside the perception range of the robot all the time are also considered.

An usual approach to this problem is that of *reactive* systems (e.g., [1]). However, basic reactive systems suffer from two shortcomings. First, they are difficult to program. Second and most important, pure reactive controllers may generate inefficient trajectories since they select the next action as a function of the current sensor readings and the robot's perception is limited.

To address this second shortcoming, some approaches combine planning and reaction (e.g., [2, 3]). In contrast to classical planning that acts on a perfect (or sufficiently good) model of the environment, these approaches only require a coarse global map of the environment made out of landmarks. Then, planning takes place at an abstract level and all the low level details are handled by the reactive component as the robot actually moves. In the case of robots operating in initially unknown environments, this global map can be built from sensory data gathered either while travelling to the goal (e.g., [2]) or in a previous exploration phase (e.g., [3]).

Global maps are a valuable aid for navigation which must be used when available. When not, we claim that the addition of learning capabilities to reactive systems is sufficient to allow a robot to generate efficient trajectories after a limited experience. This paper presents experimental results that support this claim: a real autonomous mobile robot equipped with low-resolution sensors learns efficient goal-oriented obstacle-avoidance reactive strategies in a few trials. Moreover, a learning approach like ours even overcomes the first shortcoming of reactive systems. As some researchers have recently shown, the robot programming cost is considerably reduced by letting the robot learn automatically the appropriate navigation strategies (e.g., [4, 5, 6, 7, 8]).

This paper describes the testing of a reinforcement connectionist architecture on a real autonomous mobile robot. A reinforcement-learning robot *learns by doing* and does not require a teacher who proposes correct actions for all possible situations the robot may find itself in. Instead, the robot simply tries different actions for every situation it encounters and selects the most useful ones as measured by a *reinforcement* or performance feedback signal. Reinforcement connectionist learning [9, 10, 11] brings four benefits to autonomous robots. First, this kind of learning robot can improve its performance continuously and can adapt itself to new environments. Second, the connectionist network does not need to represent explicitly

This research has been partially supported by the ESPRIT Basic Research Action number 7274.

all possible situation-action rules as it shows good generalization capabilities. Third, connectionist networks have been shown to deal well with noisy input data, a capability which is essential for any robot working upon information close to the raw sensory data. Fourth, connectionist learning rules are well suited to on-line and real-time learning.

In addition to these benefits, the architecture described in this paper also overcomes three critical limitations of basic reinforcement connectionist learning that prevent its application to autonomous robots operating in the real world. The first and most important limitation is that reinforcement learning might require an extremely long time. The main reason is that it is hard to determine rapidly promising parts of the action space where to search for suitable reactions. The second limitation has to do with the robot's behavior during learning. Practical learning robots should be operational at any moment and, most critically, they should avoid catastrophic failures such as collisions. Finally, the third limitation concerns the inability of "monolithic" connectionist networks — i.e., networks where knowledge is distributed over all the weights— to support incremental learning. In this kind of standard networks, learning a new rule (or tuning an existing one) could degrade the knowledge already acquired for other situations.

An important assumption of our approach is that the robot receives a reinforcement signal after performing every action. Although this assumption is hard to satisfy in many reinforcement learning tasks, it is not in the case of goal-directed tasks since the robot can easily evaluate its progress towards the goal at any moment. A second assumption of our approach is that the goal location is known. In particular, the goal location is specified in relative cartesian coordinates with respect to the starting location.

II. Experimental Setup

The system composed by the robot and the reinforcement connectionist controller has been called TESEO. The physical robot is a wheeled cylindrical mobile platform of the *Nomad 200* family . It has three independent motors. The first motor moves the three wheels of the robot together. The second one steers the wheels together. The third motor rotates the turret of the robot. The robot has 16 infrared sensors and 16 sonar sensors, from which distances to the nearest obstacles can be estimated, and 20 tactile sensors detect collisions. The infrared and sonar sensors are evenly placed around the perimeter of the turret. Finally, the robot has a *dead-reckoning* system that keeps track of the robot's position and orientation.

The connectionist controller maps the currently perceived situation into a *spatially continuous action*. Then, the controller waits until the robot has finished to perform the corresponding motor command before computing the associated reinforcement signal and the next action. We will describe next what the input, output and reinforcement signals are.

The input to the connectionist network consists of a vector of 40 components, all of them 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 are derived from a virtual sensor that provides the distance between the current and goal robot locations. This sensor is based on the dead-reckoning system. The 8 components correspond to a *coarse codification* of an inverse exponential function of the virtual sensor reading. The main reason for using this codification scheme is that, since it achieves a sort of interpolation, it offers three theoretical advantages, namely a greater robustness, a greater generalization ability and faster learning.

The output of the connectionist network consists of a single component that controls directly the steering motor and indirectly the translation and rotation motors. This component is a real number in the interval $[-180, 180]$ and determines the direction of travel with respect to the vector connecting the current and goal robot locations. Once the robot has steered the commanded degrees, it translates a fixed distance (10 inches) and, at the same time, it rotates its turret in order to maintain the front infrared and sonar sensors oriented toward the goal.

In this way, a *relative codification* of both sensor readings and motor commands with respect to the goal direction is always maintained. This codification scheme is directly responsible for TESEO's generalization capabilities.

The reinforcement signal 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. Thus actions incur a cost which depends on both the step clearance and the step length. Concerning the step clearance, the robot is constantly updating its sensor readings while moving. Thus the step clearance is the shortest distance measured by any of the sensors while performing the action.

Finally, TESEO is equipped with a low-level asynchronous emergency routine to prevent collisions. The robot stops and retracts whenever it detects an obstacle in front of it which is closer than a safety distance.

III. Extensions to Basic Reinforcement Learning

TESEO's aim is to learn to perform those actions that optimize the total reinforcement received along the trajectory to the goal. As mentioned in the Introduction, TESEO has been designed to overcome 3 limitations of basic reinforcement learning: slow convergence, lack of incremental improvement, and failure to be operational from the very beginning. The following three aspects of TESEO's architecture address these limitations.

First, the connectionist controller is a *modular network*, each module codifying a *set of similar reaction rules*. That is, these rules map similar sensory inputs into similar actions and, in addition, they have similar long-term consequences. Modularity guarantees that improvements in the response to a given situation will not negatively alter other unrelated reactions.

Second, 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 IV.B). This exploration technique allows TESEO to avoid experiencing irrelevant actions and to minimize the risk of collisions.

Third, instead of learning from scratch, TESEO utilizes a fixed set of *basic reflexes* every time its connectionist controller fails to generalize correctly its previous experience to the current situation. The connectionist controller associates the selected reflex with the perceived situation in one step. This new reaction rule is tuned subsequently through reinforcement learning. Basic reflexes correspond to previous elemental knowledge about the task and are codified as simple reactive behaviors [1]. Each 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. 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. *Integrating learning and reaction* in this way allows TESEO to focus on promising parts of the action space immediately and to be operational from the very beginning.

IV. Controller Architecture

The connectionist controller is a modular two-layer network (Fig. 1). The first layer consists of units with *localized receptive fields*, which we call *exemplars*, because each of them represents a point of the input space and covers a limited area around this point. The second layer is made of one single stochastic linear unit, the output unit. There exists a full connectivity between the exemplars and the output unit.

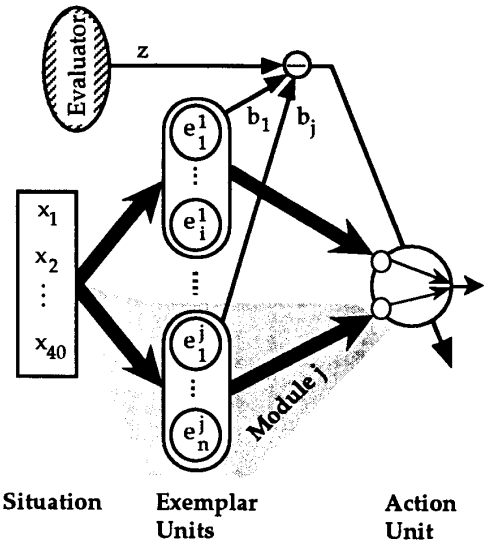

Fig. 1. Controller architecture.

A. Exemplars

The activation level of an exemplar is a value in the interval $[0, 1]$, it being 0 if the perceived situation is outside its receptive field, and 1 if the situation corresponds to the point where the exemplar is centered. The jth module consists of the exemplars $e^j_1, \ldots, e^j_n$ and their related links.

All modules respond to each perceived situation, but only the module owning the exemplar with the maximum response propagates the activities of its exemplars to the output unit. If no exemplar "matches" the perceived situation —i.e., if the input does not fall in the receptive field of any exemplar—, then the basic reflexes are triggered and the current situation becomes a new exemplar. Section V.A provides more details on this resource-allocating procedure.

The modules are not predefined, but are created dynamically as TESEO explores its environment. Every module j keeps track of four adaptive values. First, the width of the receptive fields of all the exemplars in this module, d_j. Second, the expected total future reinforcement, b_j, that the robot will receive if it uses this module for computing the next action. 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. There are as many prototypical actions as reflexes. Thus the first one is the direction of the front infrared and sonar sensors —i.e., a deviation of 0 degrees from the vector connecting the

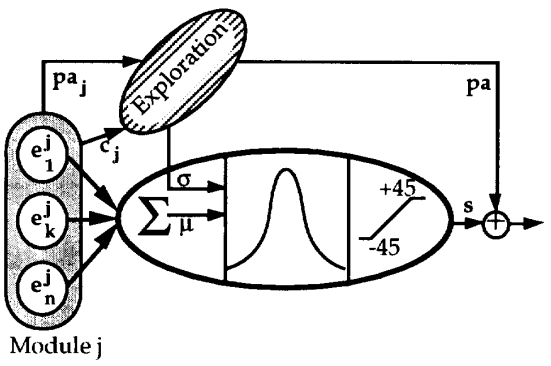

Fig. 2. The output unit.

current and goal robot locations—, the second one is 22.5 degrees, and so on. Sections V.A and VI.B explain how pa_j is initially determined and how it evolves, respectively.

After reacting, the *evaluator* computes the reinforcement signal, z, as specified in Section II. Then, if the action was computed through the module j, the difference between z and b_j is used for learning (see Section V.D). Only the weights of the links associated to the winning module j are modified.

B. Output Unit

The output of the connectionist controller is a prototypical action pa, normally the prototypical action of the winning module, plus a certain variation s that depends on the location of the perceived situation in the input subspace dominated by that module (Fig. 2).

In order to find a suitable action for each situation through reinforcement learning, TESEO needs to explore the action space. However, this exploration is not conducted upon the whole action space, but 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. This exploration mechanism depends on c_j, the counter associated to the winning module.

On the one hand, the exploration mechanism selects pa from the prototypical actions associated to all the modules that classify the perceived situation. That is, if the situation is located in the receptive field of one of the exemplars of the module m, then pa_m is a candidate. Assuming j to be the winning module, the selection goes as follows. If c_j is not divisible by, say, 3 then pa_j is chosen. Otherwise, pa is taken to be the prototypical action associated to the module m with the best expected total future reinforcement, b_m. The basic idea behind this exploration mechanism is that the winning module could well benefit from the knowledge of neighboring modules.

On the other hand, the deviation s from pa is computed through a *stochastic* process in the interval $[-45°, 45°]$. Thus, 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 $e^j_1, \ldots, e^j_n$ of the winning module:

$$\mu = \sum_{k=1}^{n} w^j_k a^j_k, \tag{1}$$

where w^j_k is the weight associated to the link between e^j_k and the output unit, and a^j_k is the activation level of e^j_k. 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:

$$l = N(\mu, \sigma). \tag{2}$$

In the third step, the unit computes s:

$$s = \begin{cases} 45, & \text{if } l > 45, \\ -45, & \text{if } l < -45, \\ l, & \text{otherwise.} \end{cases} \tag{3}$$

V. Five Learning Mechanisms

A. Network Growth

The first learning mechanism makes the controller network 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 IV.A, if no exemplar "matches" the perceived situation, then the basic reflexes are triggered and the current situation becomes a new exemplar. That is, both represent the same point of the input space. 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 the new exemplar is added to a new module. This module consists initially of the exemplar and its associated connections. Concerning the four parameters associated to this new module f, they are initially set to the following values: d_f equals 0.5, c_f equals 0, pa_f is the selected reflex, and b_f is estimated on the basis of the distance from the next location to the goal and the distance from the next location to the perceived obstacles in between the robot and the goal.

B. Tuning Exemplars

The second 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 matches the input coordinates $\mathbf{x}$:

$$\mathbf{v}_k^j(t+1) = \mathbf{v}_k^j(t) + \epsilon * a_k^j(t) * \left[\mathbf{x}(t) - \mathbf{v}_k^j(t)\right], \quad (4)$$

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

C. Improving Reinforcement Estimates

The third learning mechanism is related to the update of the future reinforcement estimates b_j, and it is based on *temporal difference (TD) methods* [12].

Every value 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[1]:

$$b_j = \max_{Actions}(z) + b_i. \quad (5)$$

In order to iteratively update the values of b_j, so that finally (5) holds for all of them, we have used the simplest TD method, i.e. TD(0) (see [12] for details). If the situation perceived at time t is classified by module j and, after performing the computed action, the next situation belongs to module i and the reinforcement signal —or cost— is $z(t+1)$, then:

$$b_j(t+1) = b_j(t) + \eta * \left[z(t+1) + b_i(t) - b_j(t)\right]. \quad (6)$$

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

[1] As described in Section II, z takes negative values. So, minimizing future cost corresponds to maximizing future reinforcement.

D. Weight Update

The fourth learning mechanism concerns weight modification and uses the classical *associative search (AS)* [9]. AS uses the estimation given by TD to update the situation-action mapping, which is codified into the connectionist controller:

$$w_k^j(t+1) = w_k^j(t) + \alpha * \left[z(t+1) + b_i(t) - b_j(t)\right] * \phi_k^j(t), \quad (7)$$

where α is the learning rate, and ϕ_k^j is the eligibility factor.

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 [11]:

$$\phi_k^j(t) = \frac{\partial ln N}{\partial w_k^j}(t) = a_k^j(t)\frac{l(t) - \mu(t)}{\sigma^2(t)}, \quad (8)$$

where N is the normal distribution function in (2). The weights w_k^j are modified more intensively in case of *reward* —i.e., when TESEO behaves better than expected— than in case of *penalty*. These two values of α 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.

VI. Four Learning Opportunities

Let us present now the four occasions in 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.

A. Unexperienced Situation

If the perceived situation is not classified into one of the existent 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.

B. 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 action unit, $pa + s$, is closer to a prototypical action other than pa_j, then pa_j becomes this new prototypical action. In case of penalty, c_j is increased by 1 and d_j is decreased by 0.1 if it is still greater than a threshold k_d.

C. 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 a threshold k_d.

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 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 V.A.

D. Reaching the Goal

Finally, whenever the goal is reached, the value b_j of every winning module j along the path to the goal is also updated in reverse chronological order. Thus, TESEO needs to store the pairs $\langle j(t), z(t+1) \rangle$ along the current path. This supplementary update only accelerates the convergence of the b_j's, but does not change their steady values [13].

VII. 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 to a point at the end of the corridor. TESEO achieves the target location every time and it never gets lost or trapped into malicious local maxima. The first time it tries to reach the goal it relies almost all the time on the basic reflexes. Essentially, the basic reflexes make TESEO travel in the direction that (*i*) is the closest to the current one, (*ii*) is the safest, and (*iii*) brings TESEO toward the goal. As illustrated in Fig. 3, 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 sensors were too large resulting in specular reflections.

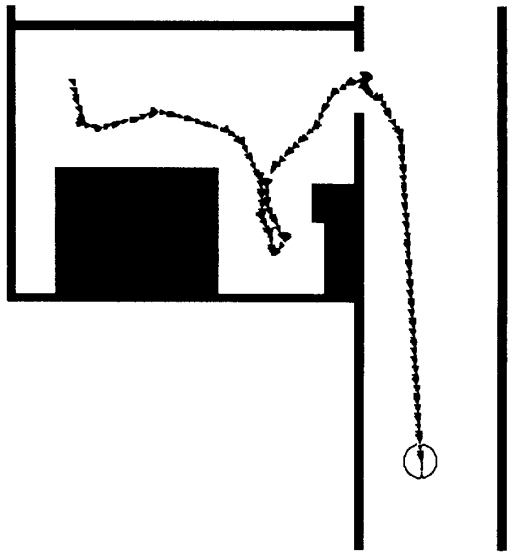

Fig. 3. The environment and first trajectory generated for a starting location within the office. Note that TESEO has some problems in going through the doorway.

Thus this simple task offers TESEO the opportunity to learn three skills. The first one is to smooth out certain sections of the trajectory generated by the basic reflexes. The second skill is to be able to avoid dead-ends or, in general, not follow wrong walls. The third and most critical is to avoid obstacles that its sensors cannot detect.

TESEO learns these three skills *very rapidly*. It reaches the goal efficiently and without colliding after travelling 10 times from the starting location to the desired goal (Fig. 4). The total length of the first trajectory is approximately 13 meters while the length of the trajectory generated after TESEO has learned the suitable sequence of reactions is about 10 meters. This experiment was run several times, obtaining similar results.

Concerning the acquisition of the third skill, TESEO associates safe actions to malicious situations —made out of a coarse codification of the distance to the goal and of uncorrect estimated distances to the obstacles— immediately after colliding with unperceived obstacles. Moreover, due to its noise tolerance and generalization capabilities (see below), TESEO only hits a few times with obstacles that its sensors cannot detect before learning to

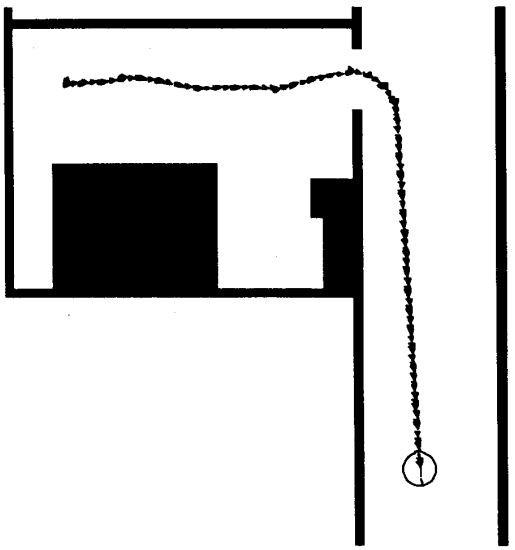

Fig. 4. Trajectory generated after travelling 10 times to the goal.

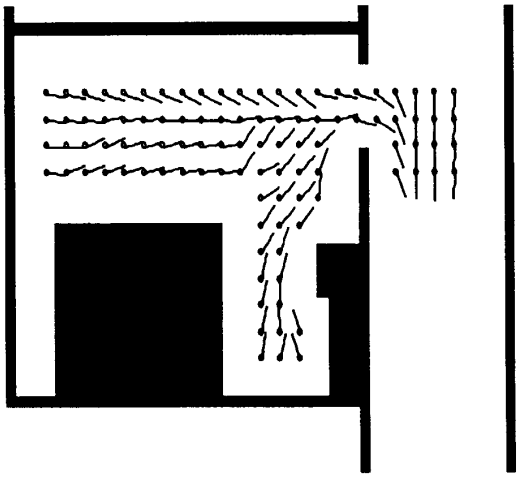

Fig. 5. Generalization abilities: Situation-action rules applied for a sample of locations within the office and the first part of the corridor.

avoid them. In addition, the learned navigation strategies have the following features. First, the trajectories are *quite smooth* even if the basic reflexes have not been programmed in this way (it is rather difficult to do it!!) and the reinforcement signal only penalizes long and unsafe paths. Second, the acquired reactions are *robust to noisy sensory data*. Since sensors are not perfect, TESEO does not perceive the same situations along similar trajectories. However, it is still able to generate suitable actions using only its acquired reactions.

Fig. 5 illustrates instances of the reaction rules learned. For every location considered (little circles) the move to be taken is depicted. Fig. 5 shows that TESEO generates solution paths from any starting location inside the room. This simulation experiment 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 obstacles are put in the way to the goal. TESEO moves around the obstacles and then it returns to the original, efficient trajectory. In other experiments, the goal 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.

VIII. Related Work

The robot learning architecture described in this paper shows some resemblances with several previous works.

In the case of goal-directed tasks, it is possible to use the future reinforcement associated with situation-action pairs to define a vector field over the environment. Then, the robot will normally perform the action that follows the gradient of the vector field. Seen in this way, our approach is reminiscent of potential field approaches (e.g., [14, 15]). However, our approach differs from them in three fundamental aspects. First, any potential field algorithm relies on predetermining a potential function that will allow the robot to reach the goal efficiently. Contrarily, a reinforcement approach like ours is *adaptive* in that it starts with an initial and inefficient vector field generated from rough estimates of the future reinforcements and adapts the field to generate efficient trajectories as it converges to the correct estimates through TD methods. Second, the kind of vector fields that can arise in our case are much more varied than those defined by a weighted sum of the potential fields originated by the different obstacles. Third, most of the potential field approaches require a global model of the environment while our approach only needs local information obtained from the robot's sensors. There exist also potential field approaches which only need local workspace models built out of sensory information (e.g., [14]), but they cannot produce efficient trajectories.

The benefits of letting the robot learn automatically the appropriate reaction rules have recently been emphasized by several authors. [4] combines reinforcement connectionist learning and teaching to reduce the learning time. In this framework, a human teacher shows the robot several instances of reactive sequences that achieve the task. Then, the robot learns new reaction rules from these

examples. The taught reaction rules help reinforcement learning by biasing the search for suitable actions toward promising parts of the action space. In our approach, the basic reflexes play the same guidance role, requiring only a programmer instead of a teacher. [5] use Kohonen maps [16] to split the sensory input space into clusters, and then associate an appropriate action to every cluster through reinforcement learning. Their architecture maps all the situations of a given cluster to a single action, and classifies situations solely by the similarity of their representations. Our architecture classifies situations, first, based on the similarity of their input representations. But, then, it also incorporates task-specific information for classifying based on the similarity of reinforcements received. In this manner, the input space is split into *consistent* clusters since similar future reinforcement corresponds to similar suitable actions for similar situations. [7] integrate inductive neural network learning and explanation-based learning. The domain theory is previously learned by a set of neural networks, one network for each discrete action the robot can perform. Our basic reflexes also represent prior knowledge about the task; however, they are much more elemental and are used in a different way.

Our approach is also related to the Dyna integrated architectures introduced by [17] and further extended by [4, 18, 19], among others. Roughly, Dyna uses planning to speed up the acquisition of reaction rules through reinforcement learning. Dyna also learns a global model of the task on which it plans.

IX. Summary and Future Work

We have described a reinforcement learning architecture that makes an autonomous mobile robot rapidly learn efficient navigation strategies in an unknown indoor environment. Our robot TESEO not only is operational from the very beginning and improves its performance with experience, but also learns to avoid collisions even when its sensors cannot detect the obstacles. This is a definite advantage over non-learning reactive robots. TESEO also exhibits incremental learning, high tolerance to noisy sensory data, and good generalization abilities. All these features make our robot learning architecture very well suited to real-world applications.

However, the current implementation of our approach suffers from one main limitation, namely it requires a reliable *odometry system* that keeps track of the robot's relative position with respect to the goal. Currently, odometry is totally based on dead-reckoning. 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 connectionist controller is still able to produce correct actions. But, dead-reckoning will probably be insufficient in more complicated missions requiring long travels and many turns.

Current work focusses on reliable odometry. The goal is designated by a modulated light beacon and the robot is equipped with sensors especially designed to detect that light. Whenever TESEO detects the goal it uses the beaconing system; otherwise, it relies on dead-reckoning.

References

[1] R.A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, pp. 14–23, 1986.

[2] D.P. Miller & M.G. Slack, "Global symbolic maps from local navigation," *Proc. of the 9th National Conf. on Artificial Intelligence*, pp. 750–755, 1991.

[3] J.H. Connell, "SSS: A hybrid architecture applied to robot navigation," *IEEE Int. Conf. on Robotics and Automation*, pp. 2719–2724, 1992.

[4] L.-J. Lin, "Programming robots using reinforcement learning and teaching," *Proc. of the 9th National Conf. on Artificial Intelligence*, pp. 781–786, 1991.

[5] K. Berns, R. Dillmann, & U. Zachmann, "Reinforcement-learning for the control of an autonomous mobile robot," *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1808–1815, 1992.

[6] J. del R. Millán & C. Torras, "A reinforcement connectionist approach to robot path finding in non-maze-like environments," *Machine Learning*, vol. 8, pp. 363–395, 1992.

[7] T.M. Mitchell & S.B. Thrun, "Explanation-based neural networks learning for robot control," In C.L. Giles, S.J. Hanson, and J.D. Cowan (eds.), *Advances in Neural Information Processing Systems 5*, pp. 287–294. San Mateo, CA: Morgan Kaufmann, 1993.

[8] J. del R. Millán, "Reinforcement learning of goal-directed obstacle-avoidance reaction strategies in an autonomous mobile robot," *Robotics and Autonomous Systems*, in press.

[9] A.G. Barto, R.S. Sutton, & C.W. Anderson, "Neuronlike elements that can solve difficult learning control problems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 13, pp. 835–846, 1983.

[10] A.G. Barto, R.S. Sutton, & C.J. Watkins, "Learning and sequential decision making," Tech. Report 89-95, Dept. of Computer and Information Science, University of Massachusetts, Amherst, 1989.

[11] R.J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[12] R.S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[13] S. Mahadevan & J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, vol. 55, pp. 311–365, 1992.

[14] J. Borenstein, & Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 19, pp. 1179–1187, 1989.

[15] J. Barraquand & J.C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, pp. 628–649, 1991.

[16] T. Kohonen, *Self-Organization and Associative Memory*. Second Edition. Berlin: Springer-Verlag, 1988.

[17] R.S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," *Proc. 7th Int. Conf. on Machine Learning*, pp. 216–224, 1990.

[18] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching", *Machine Learning*, vol. 8, pp. 293–321, 1992.

[19] J. Peng & R.J. Williams, "Efficient learning and planning within the Dyna framework," *Adaptive Behavior*, vol. 1, pp. 437–454, 1993.