Machine Learning, 49, 247–265, 2002

Continuous-Action Q-Learning

JOSÉ DEL R. MILLÁN jose.millan@jrc.it (http://sta.jrc.it/sba/staff/jose.htm)
DANIELE POSENATO daniele.posenato@jrc.it
ERIC DEDIEU eric.dedieu@jrc.it
Joint Research Centre, European Commission, 21020 Ispra (VA), Italy

Editor: Satinder Singh

Abstract. This paper presents a Q-learning method that works in continuous domains. Other characteristics of our approach are the use of an incremental topology preserving map (ITPM) to partition the input space, and the incorporation of bias to initialize the learning process. A unit of the ITPM represents a limited region of the input space and maps it onto the Q-values of M possible discrete actions. The resulting continuous action is an average of the discrete actions of the "winning unit" weighted by their Q-values. Then, TD(λ) updates the Q-values of the discrete actions according to their contribution. Units are created incrementally and their associated Q-values are initialized by means of domain knowledge. Experimental results in robotics domains show the superiority of the proposed continuous-action Q-learning over the standard discrete-action version in terms of both asymptotic performance and speed of learning. The paper also reports a comparison of discounted-reward against average-reward Q-learning in an infinite horizon robotics task.

Keywords: reinforcement learning, incremental topology preserving maps, continuous domains, real-time operation

1. Introduction

Reinforcement learning (RL) is thought to be an appropriate paradigm to acquire control policies for autonomous agents that work in initially unknown environments. Most reinforcement-based agents aim to learn a policy—i.e., a mapping from sensory situations to actions—whereby they maximize a *value function* that estimates the expected cumulative reward in the long term. Those agents use *temporal difference (TD) methods* (Sutton, 1988) to learn the value function associated to the problem they deal with. The most popular RL method is *Q-learning* (Watkins, 1989), which estimates the value of situation-action pairs, because of its simplicity and well-developed theory.

However, its standard tabular formulation is not appropriate when agents are real robots facing real-world tasks. In many real-world tasks the sensory and action spaces are continuous. Thus an a-priori set of discrete actions is unlikely to contain the optimal action for every possible situation—unless tables are so fine-grained and big that learning cannot converge in a reasonable time. In such a case, discrete-action Q-learning becomes unstable and not satisfactory. Practical learning robots require *compact representations* able to generalize experience in continuous domains.

This paper proposes a direct extension of Q-learning that generates continuous-valued actions. This method is combined with two other ones that tackle further challenges of

reinforcement-based robots. The first challenge is to represent appropriate high-dimensional sensory spaces. For this, we use a local network in the form of an *incremental topology preserving map* (*ITPM*) (Millán, 1997). This ITPM optimizes the coverage of the sensory space using adaptive resolution. A unit represents a limited region of the sensory space and memorizes the Q-value of M possible discrete actions. Units are added incrementally to limit the resources to the actual parts of the sensory space experienced by the robot. The second challenge is that real robots cannot afford long and risky learning trials. This is handled by incorporating *domain knowledge bias* (Millán, 1996). When a new unit is created, its Q-values are initialized to yield an action that is safe even though not optimal.

The network controller selects a continuous-valued action as an average of the discrete actions of the "winning unit" weighted by their Q-values. Then, TD(λ) updates the Q-values of the discrete actions according to their contribution.

Some authors have extended the simple tabular Q-learning to deal with continuous situation spaces (e.g., Watkins, 1989; Lin, 1992; Tham, 1995; Sutton, 1996) by means of function approximators. In particular, they use neural networks to generalize the value function across situations. These works, however, still assume discrete actions. Santamaría, Sutton, and Ram (1998) go further and consider continuous action spaces, but their approach cannot actually generate continuous actions—except when exploring randomly a small fraction of the time. In other words, this approach doesn't yield truly continuous-action policies.

Our continuous-action Q-learning approach is quite efficient computationally. Indeed, it allows real-time robot control and learning. The approach has been evaluated using both a real robot and its simulated counterpart in several robotics tasks, namely wall following and passing through a doorway. In this paper we will only detail results obtained in the first task for two reasons. Firstly, many behavior-based implementations of wall following work with discrete actions (e.g., Matarić, 1992). This doesn't mean that this task has a discrete nature, but allows a fair comparison of our continuous-action approach with its discrete-action counterpart. Secondly, wall following is an infinite horizon task and so well-suited for *average-reward* RL methods (e.g., Schwartz, 1993; Mahadevan, 1996; Tadepalli & Ok, 1998). We have compared them against standard *discounted-reward* RL methods on this real-world task to understand better those new average-reward methods. Experimental results show the superiority of the proposed continuous-action versions over the discrete-action versions in terms of both asymptotic performance and speed of learning. On the other hand, discrete-action average-reward methods learn faster than discrete-action discounted-reward methods. However, for the continuous-action versions of both kinds of methods the opposite is true: discounted-reward methods perform better than average-reward ones.

Given the experimental nature of this paper and its context (i.e., this special issue), readers are assumed familiar with the theory of reinforcement learning. Thus we avoid here a rigorous formal treatment as well as discussions on proofs of the asymptotic convergence of TD methods. For those, the interested reader is referred to Sutton and Barto (1998) and references therein.

The rest of the paper is organized as follows. Section 2 discusses the learning architecture, details the ITPM, and presents the general learning algorithm. Section 3 describes the different RL methods we have explored. Section 4 reports the experimental results. Finally, Section 5 draws some conclusions.

2. Incremental topology preserving map

An *incremental topology preserving map* (*ITPM*) consists of units and edges between pairs of units. Connected units are said to be neighbors. When used as a neural controller, it maps the current sensory situation **x** onto the next motor action a. This mapping is implemented as a vector of Q-values associated to M discrete actions. Units are created incrementally and incorporates bias (see Section 2.1). After being created, the units' sensory component is tuned by *self-organizing rules* (see Section 2.2). Their action component is updated through *reinforcement learning* (see Section 3). The general learning algorithm integrating all methods is detailed in Section 2.3.

2.1. Units and bias

Initially, the ITPM has no units and they are created as the robot uses *built-in reflexes* (Millán, 1996). The reflexes correspond to domain knowledge about the task and allow the incorporation of *bias* into the network. Bias suggest actions for situations that otherwise would be time consuming to learn. Thus bias accelerates the learning process since it focusses the search process on promising parts of the action space immediately. Furthermore, it makes the robot operational from the very beginning and increments the safety of the learning process.

Units in the network have *overlapping localized receptive fields*; i.e., each one represents a point of the input space and covers a limited area around it. The radius of the units' receptive field is k_r. Reflexes get control of the robot if the neural controller makes incorrect generalizations; i.e., when either the perceived situation is outside the receptive field of all the existing units[1] or when the network's action brings the robot to a failure (e.g., a collision). Reflexes generate M possible discrete motor actions. It is worth noting that, except in simple cases, it would be very difficult to program reflexes that solve a given task efficiently. Instead, we use "easy-to-program" reflexes that just provide acceptable starting points for the RL algorithm to search appropriate actions.

Whenever the robot invokes the reflexes (i.e., generalizes badly), it adds a new unit to the ITPM. The sensory component of this unit, which gives its position in the input space, are set to the current situation **x**. Its action component is initialized so that a greedy policy would select the action a suggested by the reflexes. The Q-value of the selected discrete action a is initialized to a fixed value k_q, while all the others are given random values according to a uniform distribution in $[0, k_q/2]$.

2.2. Self-organizing rules

Our ITPM is based on Fritzke's *growing neural gas* (*GNG*) (Fritzke, 1995) but differs as follows. First, GNG works off-line, while ITPM works on-line as required by autonomous learning robots. Second, GNG adds a new unit into the map after a fixed number of iterations. On the contrary, ITPM inserts units as they are needed to better cover the sensory space. Third, GNG removes a unit when it has no emanating edges,

while ITPM doesn't. For autonomous learning robots this would be hazardous since they will not perceive all sensory situations repeatedly after a fixed number of steps.

ITPM adopts the following *self-organizing rules*. Let b' and b'' be the nearest and the second nearest units for the sensory situation $\mathbf{x}'$ just perceived, respectively. If the robot invokes the reflexes, then let u be the new unit added to the ITPM. Now:

1. **IF** a new unit u has been added, **THEN**
   (A) create edges from u to b' and b'',
   (B) remove the edge between b' and b'' if there exists,
   (C) $b' \leftarrow u$;
   ELSE
   (D) create an edge between b' and b'' if they are not yet connected.
2. Move the sensory components of b' and of all its neighboring units r toward $\mathbf{x}'$:
   $\Delta \mathbf{w}_{b'} \leftarrow \delta(\mathbf{x}' - \mathbf{w}_{b'})$
   $\Delta \mathbf{w}_r \leftarrow \delta_r(\mathbf{x}' - \mathbf{w}_r)$.

where δ and δ_r are the learning rates associated to the nearest unit and their neighbors, respectively. Note that, initially, $b' = b'' = \emptyset$ and so steps $1.A$, $1.B$, and $1.D$ are not done. Also, if there is just one unit in the ITPM then $b' = b''$.

Representing a high-dimensional space by means of self-organizing rules has well-known properties (Kohonen, 1997). In addition, the main benefits of an ITPM are twofold. Firstly, it automatically allocates units in the visited parts of the input space. Secondly, it adjusts dynamically the necessary resolution in different regions. This is due to the use of units with overlapping receptive fields[2] and the insertion of new units when the current ones make big errors. For example, in the wall following task of figure 1, a new unit is added if the use of an existing unit brings the robot too close or too far from the wall. Figure 1 illustrates the variable resolution representation.

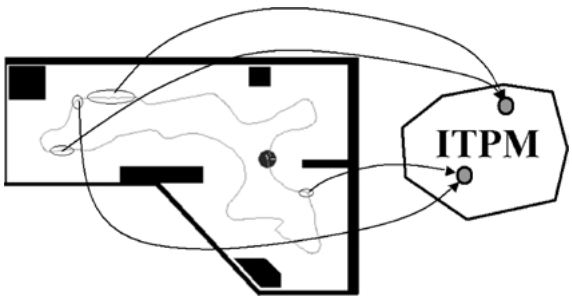

Figure 1. Example of variable resolution representation. Units of the ITPM cover regions of the sensory space and the environment whose size are different. The resolution of a region is determined by the radius of the unit's receptive field. In this case, we have recorded, for two units, the area of the environment where a unit is active while the robot performs the wall following task along particular trajectory.

As a final remark, the experiments reported later show that, in average, every unit is connected to 5 others at the end of the learning episodes. This number of neighbors is the same, independently of the RL method.

2.3. General learning algorithm

The general learning algorithm incorporating bias initialization, self-organization and reinforcement learning is:

1. *Initialization*. Perceive the initial situation $\mathbf{x}$ and compute the action a by means of the reflexes or the nearest unit b of the ITPM (see *step* 5).
2. *Loop*. Take a, perceive the next situation $\mathbf{x}'$, and get the associated reward z.
3. Find the nearest unit b' of the ITPM, if any, for $\mathbf{x}'$.
4. If $\mathbf{x}'$ is outside the receptive field of b' or is a "failure state", then
   (A) use the reflexes to compute the next action a',
   (B) add a new unit u to the ITPM centered on $\mathbf{x}'$ and with Q-values such that a greedy policy would select a', and
   (C) go to step 6.
5. Use the Q-values of the nearest unit b' to compute the next action a'.
6. *Reinforcement Learning*: adjust the Q-values of the previous nearest unit b.
7. *Self-Organization*: update the connectivity of the neighborhood of b' or u, and move the sensory component of the units.
8. $\mathbf{x} \leftarrow \mathbf{x}'$, $b \leftarrow b'$, $a \leftarrow a'$; go to step 2 if the task has not been achieved.

Next Section describes steps 5 and 6, namely the policies that work upon the Q-values of the winning unit and the different RL methods we have explored—i.e., discrete- and continuous-action, as well as discounted- and average-reward.

3. Reinforcement learning

In reinforcement learning problems, the learner attempts to acquire on-line a policy that maximizes the expected long-term reward. But, strictly speaking, this reward is only known at the end of a task—if there is any—or, at most, after the learner has been using its current policy for a while. A key element of reinforcement learning methods is the value function that estimates that expected long-term reward. Another critical component is the fact that this value function can be learned on-line by means of TD(λ) methods (Sutton, 1988). Once the learner can improve its current value function, it can also improve its current policy either through *Actor-Critic architectures* (Barto, Sutton, & Anderson, 1983) or *Q-learning* (Watkins, 1989). Both things are learned on-line as the robot performs actions that bring it from a sensory situation to the next.

In this paper we concentrate on the use of Q-learning in robotics domains characterized by continuous action spaces. Millán (1996, 1997) has extensively explored real

reinforcement-based robots facing continuous domains that adopt Actor-Critic architectures (see also Millán & Torras, 1992; Martín & Millán, 1998). In particular, our robot utilizes *on-policy TD control* methods (Sutton & Barto, 1998).

As pointed out before, the reflexes initialized the network controller (i.e., the Q-values of new units) in such a way that greedy policies are immediately operational even if far from optimal. This simplifies the *exploitation-exploration* dilemma as, for most situations, the robot needs only explore actions around the best ones currently known. In this way the robot avoids experiencing irrelevant actions and minimizes the risk of collisions. For practical purposes, however, it suffices to adopt an ϵ-greedy policy to select actions in *step* 5 of the general algorithm.

In the experiments reported later, only the nearest unit is used to select the action. Alternatively, neighboring units could also contribute to the determination of the action according to their distance from the current sensory situation. In theory, this latter technique would increase the generalization capabilities of the neural controller. But, on the other hand, it requires more computational time to select the action as well as to apply TD(λ). Moreover, this theoretical advantage tends to vanish as the Q-values of the different units converge. Because in our case learning does not start from scratch but from basic reflexes, the use of the nearest unit is enough to achieve rapid convergence and good generalization at the time that guarantees real-time robot control and learning.

3.1. Discrete-action Q-learning

If the robot uses a Q-learning method, it seeks to learn the value of the different situation-action pairs, $Q(\mathbf{x}, a)$. These values are then used to select actions so as to maximize the *discounted* sum of reward obtained while achieving the task: $\sum_{k=0}^{\infty} \gamma^k z_k$, γ being the discounted factor. After convergence, the robot follows a greedy policy and the Q-function should be

$$Q(\mathbf{x}, a) \equiv z + \gamma\, Q(\mathbf{x}', a')$$

where a' is the action the robot takes in the next situation $\mathbf{x}'$.

During learning, however, this equality does not hold and TD(λ) uses that difference as an update rule:

$$Q(\mathbf{x}, a) \leftarrow Q(\mathbf{x}, a) + \alpha(z + \gamma\, Q(\mathbf{x}', a') - Q(\mathbf{x}, a))e_{xa}$$

where α is the learning rate and e_{xa} is the eligibility factor of the situation-action pair. This factor is computed as a *replacing eligibility trace* (Singh & Sutton, 1996). Thus a unit has the following structure: $b_i = (\mathbf{w}_i, \{Q(i, \varrho)\}, \{e_{i\varrho}\} \mid \varrho = 1, \ldots, M)$. This structure is common to all the RL methods we have explored, which differ only in how the Q-values are updated and in the action selection mechanism.

The discrete-action Q-learning method works as follows. Assuming that unit i is the nearest to the situation $\mathbf{x}$, action a is selected according to an ϵ-greedy policy that chooses most of the time the discrete action with the highest Q-value. The next situation $\mathbf{x}'$ is

classified by unit i', the reward is z, and a' is the action to be taken next. The nearest i corresponds to b, while i' is either u (if a new unit has been added to the ITPM in step 4) or b'. Now *step* 6 of the general algorithm is:

6.1 Update all the replacing eligibility traces:

$$e_{\iota\varrho} \leftarrow \begin{cases} 1, & \text{if } \iota = i \text{ and } \varrho = a, \\ \lambda\gamma e_{\iota\varrho}, & \text{otherwise.} \end{cases}$$

6.2 Update all the Q-values:

$$\Delta Q(\iota, \varrho) \leftarrow \alpha(z + \gamma Q(i', a') - Q(i, a))e_{\iota\varrho}.$$

When a new unit b' is created after the reflexes suggest action a', the eligibility traces of this unit are initialized as follows: $e_{b'a'} = 1$; $e_{b'\varrho} = 0, \forall \varrho \neq a'$.

3.2. Continuous-action Q-learning

The continuous-action Q-learning method also relies upon a discrete number of actions and their corresponding Q-values. The resulting continuous action is an average of the discrete actions of the nearest unit weighted by their Q-values. This is done as follows. The discrete actions are ordered increasingly; i.e., in the case of controlling the direction of movement of a mobile robot, these actions correspond to steering commands from totally left to totally right. Assume that unit i is the nearest to the situation $\mathbf{x}$ and that action a_l is its discrete action with the highest Q-value, $Q(i, l)$. Neighboring actions to the left and right of a_l are then weighted by the difference between their Q-values and the highest one. For the sake of simplicity, we only consider one action to each side, namely a_{l-1} and a_{l+1}:

$$\begin{aligned} \mathit{left} &= \frac{1}{2 + (Q(i, l) - Q(i, l-1))^2}, \\ \mathit{right} &= \frac{1}{2 + (Q(i, l) - Q(i, l+1))^2}, \\ a &= a_l + \mathit{right} * (a_{l+1} - a_l) + \mathit{left} * (a_{l-1} - a_l). \end{aligned}$$

In this way the robot is constantly exploring actions around the currently optimal discrete one. The range of exploration depends on how good are the neighboring actions and, at most, is half the distance between two consecutive actions (hence the factor 2 in the denominators). The Q-value of the selected action a is:

$$Q(i, a) = \frac{Q(i, l) + \mathit{right} * Q(i, l+1) + \mathit{left} * Q(i, l-1)}{1 + \mathit{right} + \mathit{left}}.$$

Of course, there exist alternative ways to compute the contribution of neighboring units as well as to select continuous-valued actions. For instance, we have evaluated exponential

functions for the contributions and Gaussian processes for action selection. All the methods provide the same qualitative learning results, and so we report here one of the simplest ones.

Finally, the method updates the eligibility factors and the Q-values of the discrete actions according to their contribution. This is then *step* 6 of the general algorithm:

6.1 Update all the replacing eligibility traces:

$$e_{\iota\varrho} \leftarrow \begin{cases} \dfrac{1}{1 + right + left}, & \text{if } \iota = i \text{ and } \varrho = l, \\ \dfrac{right}{1 + right + left}, & \text{if } \iota = i \text{ and } \varrho = l + 1, \\ \dfrac{left}{1 + right + left}, & \text{if } \iota = i \text{ and } \varrho = l - 1, \\ \lambda\gamma e_{\iota\varrho}, & \text{otherwise.} \end{cases}$$

6.2 Update all the Q-values:

$$\Delta Q(\iota, \varrho) \leftarrow \alpha(z + \gamma Q(i', a') - Q(i, a))e_{\iota\varrho}.$$

3.3. Average-reward reinforcement learning

For infinite horizon tasks (e.g., a robot learning to follow walls), some authors have recently argued that *average-reward* RL methods should be more appropriate than discounted-reward ones (e.g., Mahadevan, 1996; Tadepalli & Ok, 1998). In these methods the learner seeks to maximize the average reward per action: $\lim_{n\to\infty} 1/n \sum_{t=1}^{n} z_t$.

Among the different average-reward RL methods, we have explored *R-learning* (Schwartz, 1993; Mahadevan, 1996). Tadepalli and Ok (1998) have also described a method called H-learning that, however, makes unrealistic assumptions for autonomous robots.

The discrete-action average-reward method works as follows. As for Q-learning, assume that unit i is the nearest to the situation $\mathbf{x}$, action a is selected according to an ϵ-greedy policy that chooses most of the time the discrete action with the highest Q-value, the next situation $\mathbf{x}'$ is classified by unit i', the reward is z, and a' is the action to be taken next. Now *step* 6 of the general algorithm is:

6.1 Update all the replacing eligibility traces:

$$e_{\iota\varrho} \leftarrow \begin{cases} 1, & \text{if } \iota = i \text{ and } \varrho = a, \\ \lambda\gamma e_{\iota\varrho}, & \text{otherwise.} \end{cases}$$

6.2 Update all the Q-values:

$$\Delta Q(\iota, \varrho) \leftarrow \alpha(z - \rho + Q(i', a') - Q(i, a))e_{\iota\varrho}.$$

6.3 If $Q(i, a) = \max_{\varrho} Q(i, \varrho)$, then:

$$\Delta\rho \leftarrow \beta(z - \rho + Q(i', a') - Q(i, a)).$$

ρ keeps track of the estimated average reward and it is initialized to zero. Concerning the learning rate β, it decays over time as follows:

$$\beta \leftarrow \frac{\beta}{1 + \beta}.$$

The initial value of β is β_0.

The continuous-action average-reward method works like its discounted-reward counterpart.

4. Experimental results

The continuous-action Q-learning approach has been evaluated in a real robot that must learn to follow walls. The robot is a Nomad 200 (figure 2(a)) equipped with two rings of range sensors (16 sonars and 16 infrareds) and 20 tactile sensors. The neural controller sends steering commands to the wheel motors. The input to the neural controller is derived from a *local occupancy grid*[3] (Dedieu & Millán, 1998). An occupancy grid provides more reliable information than raw sensory readings and integrate naturally different kinds of sensory signals. Figure 2(b) shows a grid built in a corridor while the robot was following one of the walls. The grid is divided into 16 sectors and, for each sector i, the distance $dist_i$ to the nearest occupied cell is computed. This distance is then tranformed according

(a) (b)

Figure 2. (a) TESEO, a Nomad 200 mobile robot, while following a wall in a corridor. It is moving toward the end of the corridor. (b) The corresponding local grid. The darker a cell, the higher is its occupancy value. Lines in a cell indicate its dominant orientation, but they are not considered in this learning task.

to the following expression: $d_i = \frac{1}{1+dist_i} \in [0, 1]$. The vector $[d_1, \ldots, d_{16}]$ is the input to the ITPM.

In order for the robot to follow a wall it must move as straight as possible while staying in between a maximum distance, r_+, and a minimum one, r_-, from that wall. The distance to the wall being followed is simply $r = \min_i dist_i$. The robot receives a reward after performing every action a. The reward function depends on this action and the next situation:

$$z = \begin{cases} 0.1, & \text{if } (r_- < r < r_+) \wedge (a \in [-5^\circ, +5^\circ]), \\ -3.0, & \text{if } (r \leq r_-) \vee (r_+ \leq r), \\ 0.0, & \text{otherwise.} \end{cases}$$

If an action brings the robot outside the "lane" $[r_-, r_+]$, then the robot stops, moves back to return inside, receives the punishment, and invokes the reflexes.

The reflexes are codified as simple reactive behaviors, namely `avoid-obstacle` (moves in the opposite direction of perceived obstacles) and `track-boundary` (steers so as not to leave the "lane"). These elemental behaviors are inspired by those of Matarić (1992). If the robot only uses the reflexes, it actually can follow walls, but along inefficient trajectories.

The continuous-action Q-learning approach makes the real robot learn rapidly to follow walls along smooth trajectories. But, we cannot afford to rely on the real robot to carry out a systematic comparison of the different RL methods described before. For this reason, the comparative study has been done with the simulated version of the Nomad 200. The simulator has a set of parameters that can be adjusted to model roughly the real robot's motion and sensors in a given environment.[4]

Table 1 provides all the parameter values used in the experiments. These values gave the best performances for the different algorithms in an initial exploratory phase (but we didn't search the parameter space exhaustively).

Table 1. Parameter values used in the experiments.

Parameter	Description	Value
k_r	Width of receptive field	0.2
k_q	Initial Q-value	3.0
δ	ITPM learning rate, nearest unit	0.05
δ_r	ITPM learning rate, neighbors	0.005
ϵ	Exploration rate of discrete methods	0.0
λ	Trace-decay factor	0.7
γ	Discounted factor	0.95
α	TD learning rate	0.1
β_0	Average reward rate, initial value	0.1
M	Number of discrete actions	$5 : -30^\circ, -15^\circ, 0^\circ, 15^\circ, 30^\circ$
r_-	Minimum distance to wall	50 cm
r_+	Maximum distance to wall	110 cm

The performance of the different RL methods is evaluated every episode of 250 control steps according to three criteria. First, *failures* corresponds to the total number of steps the robot has left the "lane". Second, the *reward* accumulated. Third, *steering* indicates the total degrees the robot steers. This amount is shown with a resolution of $1/10$ of degree and gives an idea of the smoothness of trajectories. The more straight and smooth a trajectory, the better is the learned behavior. Finally, we have also reported the number of *units* created every episode.

In the sequel we compare pairs of RL methods. The figures show mean values over 15 runs. We have checked the significance of the difference of final performances by means of a Student's t test.

In order to appreciate how much the robot learns with the different RL methods, the performance of the robot when it only relies on the built-in reflexes is, in average: 7 failures, -11.5 of reward, and 1220° of steer per episode.

Figure 3 compares the performances of the two discounted-reward Q-learning methods, namely discrete- and continuous-action. Concerning failures and reward, the superiority of

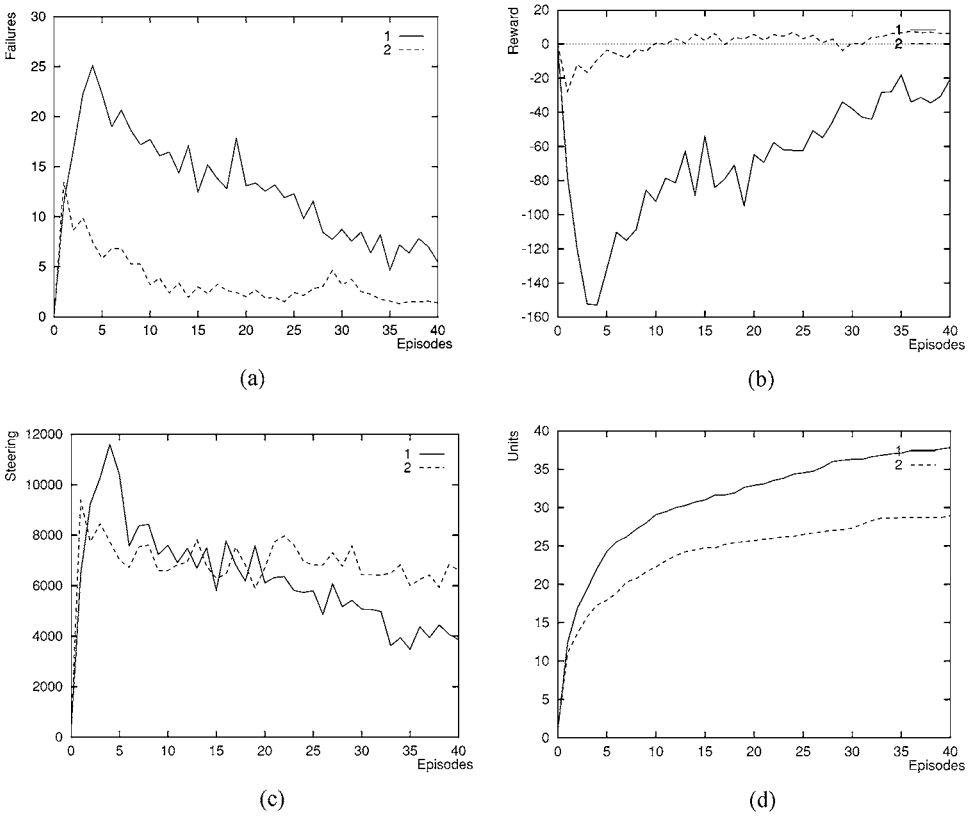

Figure 3. Performance comparison of the two discounted-reward Q-learning methods. (1) discrete-action, (2) continuous-action.

the proposed continuous-action Q-learning over the standard discrete-action version is clear in terms of both asymptotic performance and speed of learning. The continuous version outperforms the discrete one from the very beginning. As for steer, the continuous-action method makes the robot steer slightly over 1° more per step than the discrete-action one. This difference, which cannot be appreciated while observing the behavior of the two robots, is due to the exploratory nature of the continuous version. Finally, both methods create approximately the same number of units. This suggests that they explore the same regions of the sensory space. The difference is due to the fact that the discrete version leaves the "lane" more frequently, and whenever the robot does it a new unit is created. It is worth noting that most units are added at the early stages of learning. Later on, the robot simply tunes the initial reflexes and adapts to the environment it is working in. Differences in final performance (failures, reward and steer) are statistically highly significant ($p < 0.001$).

The reasons why the continuous-action method outperforms the discrete-action method are twofold. Firstly, even though wall following has been extensively handled with discrete actions, the optimal action for every possible situation is most likely continuous. This explains the fact that the continuous method converges to better performance values. It is worth noting also that our simple continuous-action Q-learning method achieves this theoretical advantage very rapidly. Secondly, the discrete method uses the TD errors to modify only one Q-value per unit at each step. This is the Q-value associated to the discrete action the robot selected when that unit was active. On the other hand, the continuous method changes three Q-values, namely those associated to the three discrete actions involved in the selection of the final action. Furthermore, the continuous method not only updates more Q-values per unit at every step, but also updates its policy instantaneously to try to select better actions. Thus, the continuous method benefits immediately of even small TD errors, while the discrete method will require several negative TD errors to make the wrongly highest Q-value of a unit decrease below the Q-value associated to a better action.

Figure 4 compares the performances of the two average-reward Q-learning methods. Here the difference between the continuous- and the discrete-action methods is not so dramatic. But the continuous version performs better than the discrete one since the beginning with respect to all the paramaters (failure, reward, and steer). Also, the differences in final performance are statistically highly significant ($p < 0.001$).

Figure 5 compares the performances of the two discrete-action Q-learning methods. As expected, average-reward Q-learning performs better than discounted-reward Q-learning with respect to both failures and reward. However, the average-reward method seems not to catch an important aspect of the learning task that is not explicitely incorporated into the reinforcement function, namely moving along straight and smooth trajectories. Indeed, as for steer the average-reward method performs at a constant level, while the discounted-reward method improves linearly over time. Figure 6 illustrates this point. Panel (a) shows a trajectory generated by the average-reward method at the end of learning. It can be observed that the learned behavior stays within the predefined "lane" but makes sharp turns. On the contrary, panel (b) depicts a trajectory obtained with the discounted-reward method. Even if not shown, this robot leaves the "lane" at several points. However, note the long straight segments that this robot follows. Also, while the robot makes oscillatory movements in

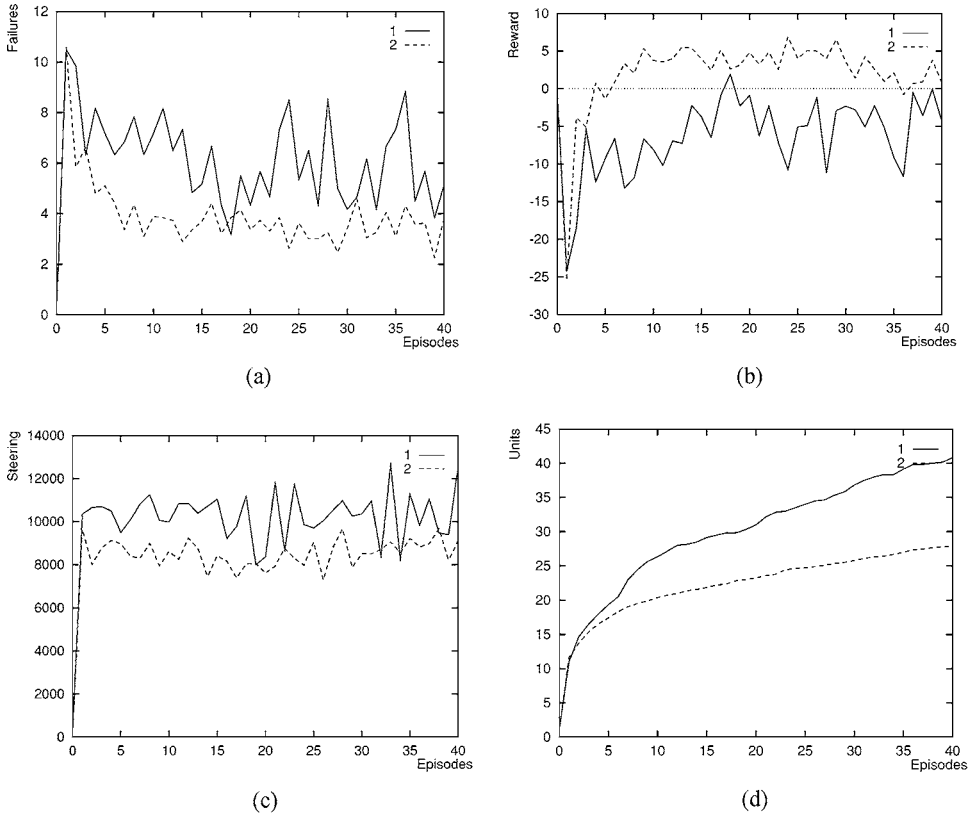

Figure 4. Performance comparison of the two average-reward Q-learning methods. (1) discrete-action, (2) continuous-action.

the upper-left part of the environment, it still moves smoothly. The differences in final performance are statistically highly significant ($p < 0.001$).

Figure 7 compares the performances of the two continuous-action Q-learning methods. Surprisingly, in this case discounted-reward Q-learning performs slightly better than average-reward Q-learning with respect to all performance criteria. Even though the average-reward method does better initially, the discounted-reward version exceedes it after around 1,000 learning steps. As in the previous cases, the differences in final performance are statistically highly significant ($p < 0.001$).

Why does not continuous-action average-reward Q-learning perform better? As Mahadevan (1996) has pointed out, average-reward Q-learning is very sensitive to exploration. His best results were obtained with high degrees of exploration. However, a key feature of our RL approach is to concentrate exploration around the best actions currently known as learning doesn't start from scratch. It may happen that the continuous-action average-reward is not exploring sufficiently. But increasing the amount of exploration in continuous action spaces will slow down convergence dramatically.

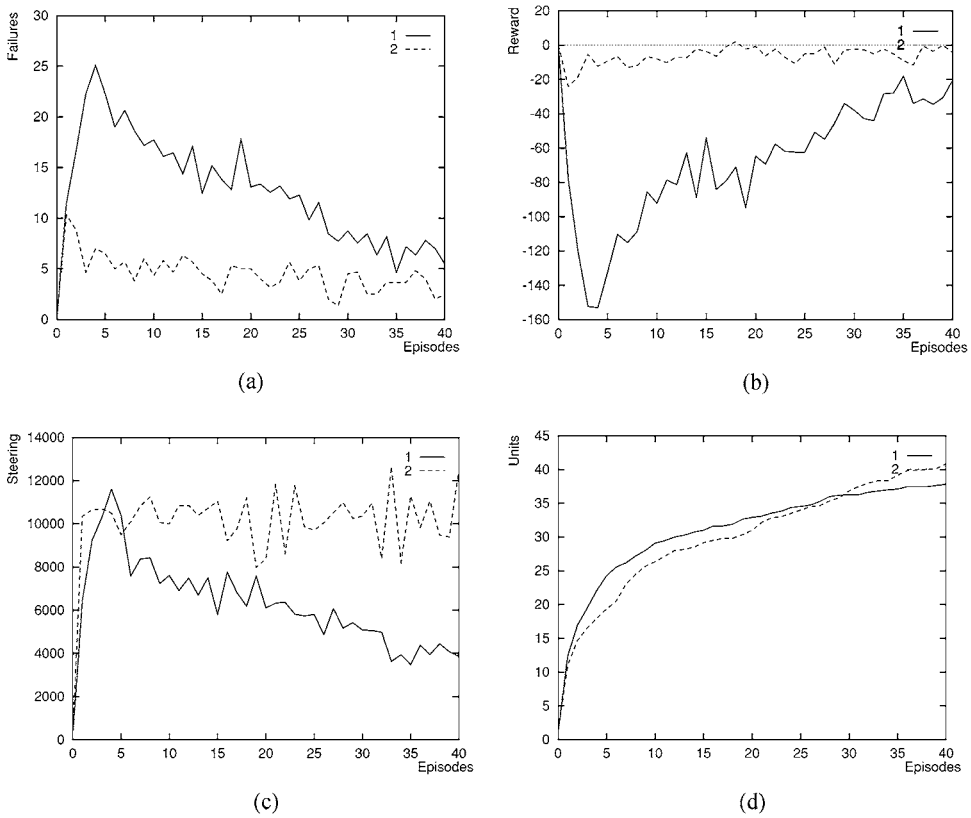

Figure 5. Performance comparison of the two discrete-action Q-learning methods. (1) discounted-reward, (2) average-reward.

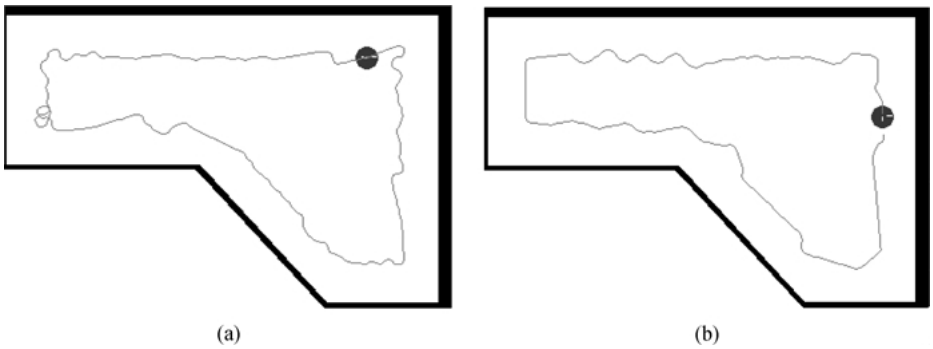

Figure 6. Examples of trajectories generated by the discrete-action Q-learning versions of average-reward (a) and discounted-reward (b) after 10,000 learning steps. Both robots move in clockwise direction.

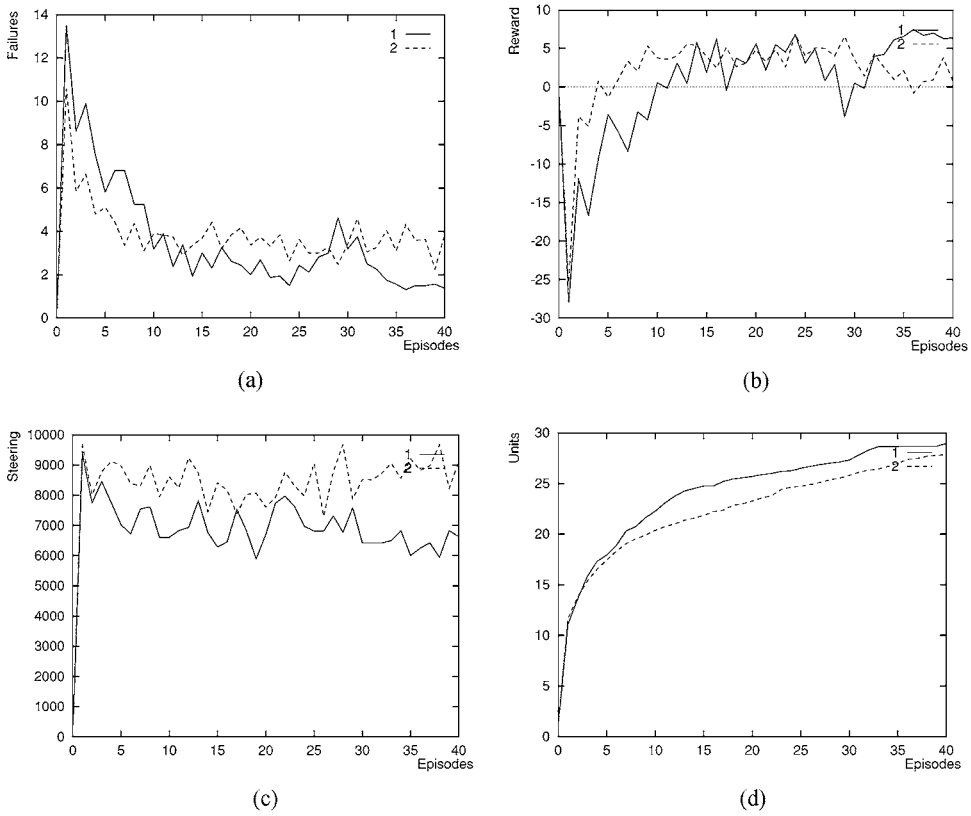

Figure 7. Performance comparison of the two continuous-action Q-learning methods. (1) discounted-reward, (2) average-reward.

We have also tested the performance of the learned navigation strategies in an environment different from that used during training. This new environment is the same room as before that, however, contains shelves and tables (see figure 8). In this experiment the robot uses continuous-action discounted-reward Q-learning as it performs the best, and it is more simple and general than the average-reward counterpart. Figure 8 shows a trajectory generated by the reinforcement-based robot in this new environment after a re-training period of 10,000 steps. In this period, the robot only adds a few more units to deal with new regions of the sensory space and, essentially, adapts quickly its previous knowledge to the new environment. Figure 9 illustrates the performance (mean values over 15 runs) of this robot during re-training. Notice how the navigation strategies acquired in the original environment are not completely satisfactory for this new one, but the robot improves its performance quite rapidly. In fact, it has learned appropriate reactive rules after 5,000 steps and later on its performance stabilizes. It is worth noting that, due to the complexity of this room, the robot requires approximately 25,000 learning steps if it is trained directly in it, while it takes approximately 15,000 learning steps if is trained first in the previous

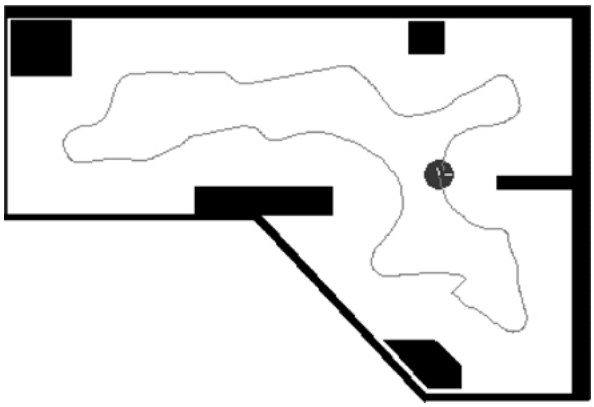

Figure 8. Example of trajectory generated by continuous-action discounted-reward Q-learning in a new environment after a re-training period.

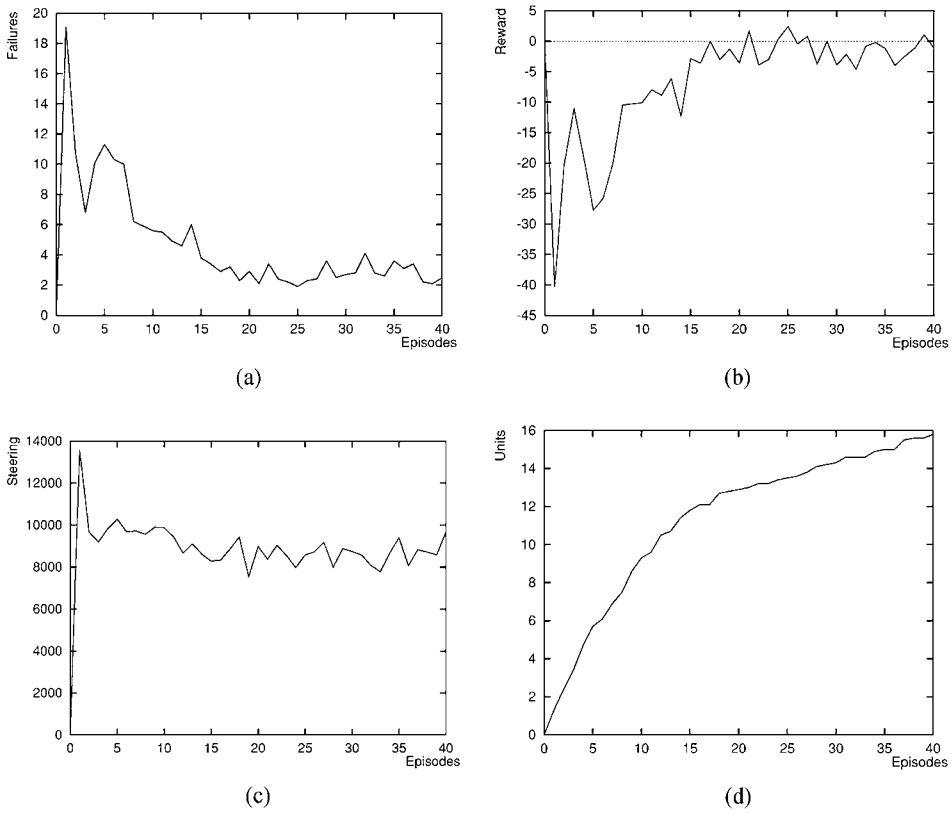

Figure 9. Performance of continuous-action discounted-reward Q-learning during re-training in the environment of figure 8.

simple environment. Discussing and analyzing this *shaping* effect is beyond the scope of this paper.

Finally, we want to highlight that none of the robots collide during the experiments at any stage of learning. This is an important requirement for practical reinforcement-based robots, and it is achieved by the incorporation of the reflexes that make the robot operational from the very beginning of learning.

5. Conclusions

In this paper we have presented a simple Q-learning method that works in continuous domains. It represents continuous input spaces by means of an incremental topology preserving map. It generates continuous-valued actions using the same data structures as the standard discrete-action Q-learning. As a consequence, our approach is quite efficient computationally. Indeed, it allows real-time robot control and learning. In addition, our approach integrates several methods, namely reinforcement learning, self-organizing principles and the incorporation of bias. The incremental topology preserving map is a kind of memory-based function approximator that yields a sparse coding representation. It generalizes experience to unseen regions of the continuous situation and action spaces based on units covering the visited parts of the situation space. Millán (1992) and Santamaría, Sutton, and Ram (1998) show experimentally in robotics domains the superiority of memory-based approaches over global function approximators (such as feedforward neural networks) and CMAC function approximators (that use static, uniformly distributed resources), respectively. Bias represents domain knowledge in the form of built-in reflexes. Appropriate domain knowledge is always available (and its corresponding reflexes can be easily derived) independently of the robotics task and the sensory modality. Altogether the proposed approach addresses two important requirements of practical learning robots: rapid and safe adaptation processes.

Here we have only considered ϵ-greedy exploration to focus the comparison of the different RL methods on the relevant aspects. This simple method, however, could make the learner get stuck on locally optimal actions. Because the reflexes are assumed to be good, the robot does a limited exploration around the best actions currently known. But if, for a given situation, the reflex and the optimal action are far apart, then this kind of exploration may eventually fail. In practice, our real robot uses active methods to explore the action space (Thrun, 1992).

The scope of this paper is restricted to the generation of continuous actions by means of Q-learning. There are other RL algorithms for handling continuous actions, but almost all of them are based on Actor-Critic architectures. A discussion of these methods is beyond the scope of this paper. Baird (1995) has proposed a class of RL methods called residual algorithms with guaranteed convergence for arbitrary, differentiable function approximators. A special case of this algorithm is residual Q-learning that can yield continuous actions. However, this approach requires to find the action with maximum Q value, what can be quite difficult with most function approximators. An interesting avenue of future research is to derive a residual algorithm for our architecture, which would combine guaranteed convergence with simplicity and rapid learning.

Experimental evaluation in a robotics task has shown the superiority of the proposed continuous-action Q-learning over the standard discrete-action version in terms of both asymptotic performance and speed of learning. We have also compared discounted-reward against average-reward Q-learning in an infinite horizon robotics task. It seems that average-reward RL is not always superior to discounted-reward RL. This is at least the case for a robotic task characterized by continuous domains and the continuous-action Q-learning approach described in this paper.

Notes

1. That is, when the distance of the perceived situation $\mathbf{x}$ to the nearest unit is greater than k_r. As customary, we use the Euclidean distance.
2. In this paper we will actually use an implementation of the ITPM where the units have fixed receptive fields. This simplification facilitates the comparison of different RL algorithms as it helps to isolate the relevant variables. Millán (1996) proposes a mechanism that adjusts individually each unit's receptive field, what further promotes a variable resolution representation of the sensory space.
3. Occupancy grids are spatial representations that assign to each cell of a square grid an "occupancy value" between 0 and 1, interpreted as some confidence measure of this cell not being free space. The grid moves with the robot and covers a limited area around the robot. The grid is updated by temporally integrating new raw sensory readings. Our method solves several shortcomings of grid-based techniques that arise from their discrete nature and updates grids considerably faster than previous approaches. In our case the grid has $33 * 33$ cells, each measuring $15 * 15$ cm.
4. Even though the simulator is always a simplified model of reality, it is still a valuable tool for studying robot learning approaches. Indeed, we use to rely upon the simulated robot to learn initial behaviors that are then quickly tuned on the real robot.

References

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning* (pp. 30–37).

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, 13*, 835–846.

Dedieu, E., & Millán, J. del R. (1998). Efficient occupancy grids for variable resolution map building. In *Proceedings of the 6th International Symposium on Intelligent Robotic Systems* (pp. 195–203).

Fritzke, B. (1995). A growing neural gas network learns topologies. In *Advances in neural information processing systems 7* (pp. 625–632).

Kohonen, T. (1997). *Self-organizing maps* (2nd edn.). Berlin: Springer-Verlag.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning, 8*, 293–321.

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning, 22*, 159–195.

Martín, P., & Millán, J. del R. (1998). Learning reaching strategies through reinforcement for a sensor-based manipulator. *Neural Networks, 11*, 359–376.

Matarić, M. J. (1992). Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation, 8*, 304–312.

Millán, J. del R. (1992). A reinforcement connectionist learning approach to robot path finding. Ph.D. Thesis, Software Dept., Universitat Politècnica de Catalunya, Barcelona, Spain.

Millán, J. del R. (1996). Rapid, safe, and incremental learning of navigation strategies. *IEEE Transactions on Systems, Man, and Cybernetics—Part B, 26*, 408–420.

Millán, J. del R. (1997). Incremental acquisition of local networks for the control of autonomous robots. In *Proceedings of the 7th International Conference on Artificial Neural Networks* (pp. 739–744).

Millán, J. del R., & Torras, C. (1992). A reinforcement connectionist approach to robot path finding in non-maze-like environments. *Machine Learning, 8*, 363–395.

Santamaría, J. C., Sutton, R. S., & Ram, A. (1998). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior, 6*, 163–217.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the 10th International Conference on Machine Learning* (pp. 298–305).

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning, 22*, 123–158.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, 3*, 9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems 8* (pp. 1038–1044).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tadepalli, P., & Ok, D. (1998). Model-based average reward reinforcement learning. *Artificial Intelligence, 100*, 177–224.

Tham, C. L. (1995). Reinforcement learning of multiple tasks using a hierarchical CMAC architecture. *Robotics and Autonomous Systems, 15*, 247–274.

Thrun, S. B. (1992). The role of exploration in learning control. In D. A. White & D. A. Sofge (Eds.), *Handbook of intelligent control: Neural, fuzzy and adaptive approaches* (pp. 527–559). New York: Van Nostrand Reinhold.

Watkins, C. J. C. H. (1989). Learning with delayed rewards. Ph.D. Thesis, Cambridge University, England, UK.

Received March 18, 1999
Revised March 20, 2000
Accepted December 29, 2000
Final manuscript December 30, 2000