Neural reinforcement learning for behaviour synthesis

Claude Touzet

C. TOUZET, "Neural Reinforcement Learning for Behaviour Synthesis, " to appear in *Robotics and Autonomous Systems*, Special issue on Learning Robot: the New Wave, N. Sharkey Guest Editor, 1997.

Neural Reinforcement Learning for Behaviour Synthesis

Claude Touzet
DIAM - IUSPIM
University of Aix-Marseille III
F - 13397 Marseille cedex 20, France
Email: Claude.Touzet@iuspim.u-3mrs.fr

ABSTRACT

We present the results of a research aimed at improving the Q-learning method through the use of artificial neural networks. Neural implementations are interesting due to their generalisation ability. Two implementations are proposed: one with a competitive multilayer perceptron and the other with a self-organising map. Results obtained on a task of learning an obstacle avoidance behaviour for the mobile miniature robot Khepera show that this last implementation is very effective, learning more than 40 times faster than the basic Q-learning implementation. These neural implementations are also compared with several Q-learning enhancements, like the Q-learning with Hamming distance, Q-learning with statistical clustering and Dyna-Q.

Key Words: Neural Q-learning, reinforcement learning, obstacle avoidance behaviour, self-organising map, autonomous robotics.

INTRODUCTION

In this paper, we present the results of research aimed at improving reinforcement learning through the use of artificial neural network implementations. Reinforcement learning proposes the synthesis of robot behaviours with the help of reinforcement functions. A reinforcement function uses a measure of the robot's performance to guide the learning. It is no longer necessary to write an algorithm generating the corresponding behaviours, something particularly interesting when such algorithms are not available (too complex, too expensive, etc.). Moreover, the synthesised behaviour integrates the

local performance (i.e., the heterogeneity) of the sensors, the motors, the noise of the environment, etc.

Reinforcement learning applications imply the mapping of situations to actions in huge situation-action space. On the other hand, the duration of the learning phase must be as short as possible to reduce engineering costs of development. Therefore, generalisation is of major importance in the process. Neural network generalisation, as shown by connectionist industrial developments, is very attractive. In this paper, we review neural implementations of Q-learning and also propose two new implementations.

In section 1, the real mobile robot Khepera and the environment of our experiments are described. This section also presents the reinforcement learning function used for the synthesis of an obstacle avoidance behaviour. The following section 2 reviews the Q-learning method, certainly the most used reinforcement learning method. Section 3 points out the credit assignment problem of the Q-learning method and briefly describes several enhancements. Section 4 introduces the first neural network implementation of the Q-learning, pointing out its advantages and limitations. Section 5 presents our proposal for a multilayer perceptron implementation of Q-learning. In section 6, a second neural network implementation is proposed using a self-organising map. Comparisons between the different implementations of Q-learning are presented in section 7. Results are discussed in section 8. Concluding remarks are given in section 9.

1. THE MINIATURE MOBILE ROBOT KHEPERA

Khepera is a miniature mobile robot [1] having a diameter of 6 cm and a weight of 86 g (figure 1). Two continuously independently controllable wheels allow the robot to move around. Eight infra-red sensors help the robot to perceive its environment. Their detection range is between 5 and 2 cm. Sensor data are real values between 0.0 (nothing in front) and 1.0 (obstacle nearby), each coded in 10 bits. All the measurements depend significantly on various factors like the distance from the obstacle, the colour, the reflectance, the vertical position, etc. The heterogeneity between sensors is important (figure 2) and reflects the sensor manufacturing process. The computational power of the robot is equivalent to a M68331. Energy autonomy is thirty minutes.

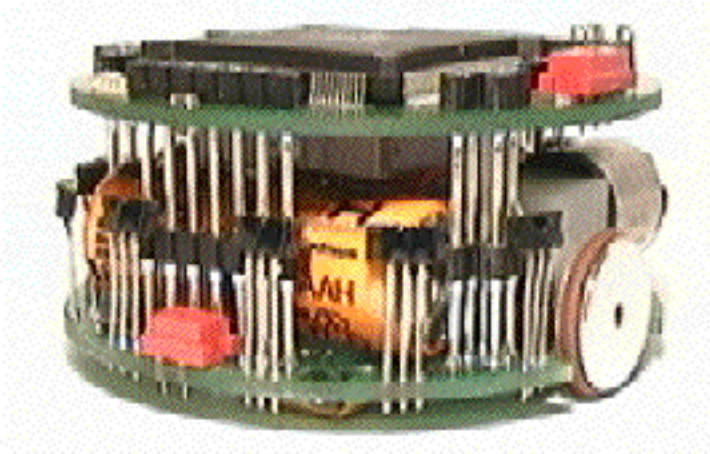


Figure 1. The miniature mobile robot Khepera.

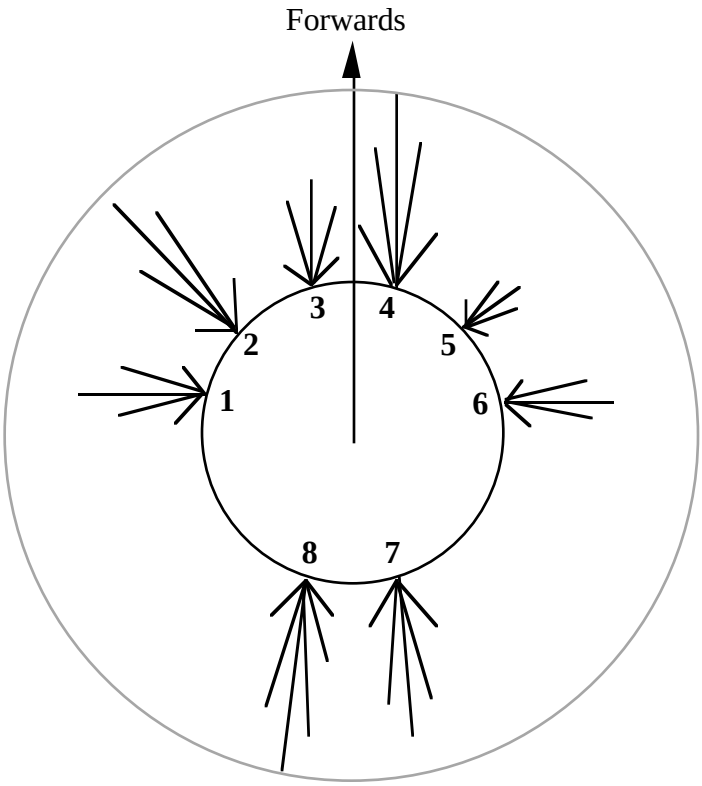


Figure 2. Maximum value of the sensor response associated to a small obstacle (a pen) rotating around the robot at a constant distance of 2 cm. The sensor value is drawn as a line segment starting at the sensor location and oriented to the obstacle. The length of the segment represents the measured sensor value (the grey circle represents a value of 1.0 which is only achieved by sensor 4 and 8). The obstacle right in front of sensor 4 generates a response value of 1.0, but the same obstacle in front of sensor 5 generates a response value of 0.4. Sensor 4 is then two times more effective than sensor 5.

1.1 Task, environment and indicator

Task

The task Khepera has to perform is to move forward when possible and avoid obstacles. Two behaviours are involved. One, with a higher priority, moves forward. A second proposes avoidance actions. The first behaviour is so simple, i.e., move forward when nothing is detected by the sensors, that it is of no interest here. However, the second behaviour involves knowing how much to turn and in which direction so as to avoid the obstacles. Our goal in dealing with this particular task is not to solve a problem for which solutions were proposed years ago [2], but to work with a problem just complex enough to be not trivial, but also simple enough to allow a good understanding of the implications of neural implementations. The Braitenberg vehicle [3] described in the commercial package of Khepera will serve us as a benchmark for our experiments. The avoidance algorithm used is described in figure 3.

Let I be the sensor vector, $I = \{i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8\}$,
let A be the speed vector, $A = \{a_l, a_r\}$

$$a_r = c_1 . i_1 + c_2 . i_2 + c_3 . i_3 + 10$$

$$a_l = c_3 . i_4 + c_2 . i_5 + c_1 . i_6 + 10$$

where c_1, c_2 and c_3 are negative constants. We choose for these constants the value given by the commercial package: $c_1 = -17$, $c_2 = -11$, $c_3 = -7$

Figure 3. The obstacle avoidance used by the Braitenberg vehicle that will serve as benchmark for our experiments.

Environment

The environment is an arena the size of a sheet of A4 paper. Obstacles are film cans or whatever is suitable (lipsticks, erasers, etc.). Obstacles are put in the arena at the beginning of the experiment (figure 4). Figure 5 shows a trace of the Braitenberg avoidance behaviour of Khepera in the arena.

Figure 4. One of the environments of our experiments with the miniature robot Khepera in the centre. Obstacle are put in the arena at the beginning of each experiment.

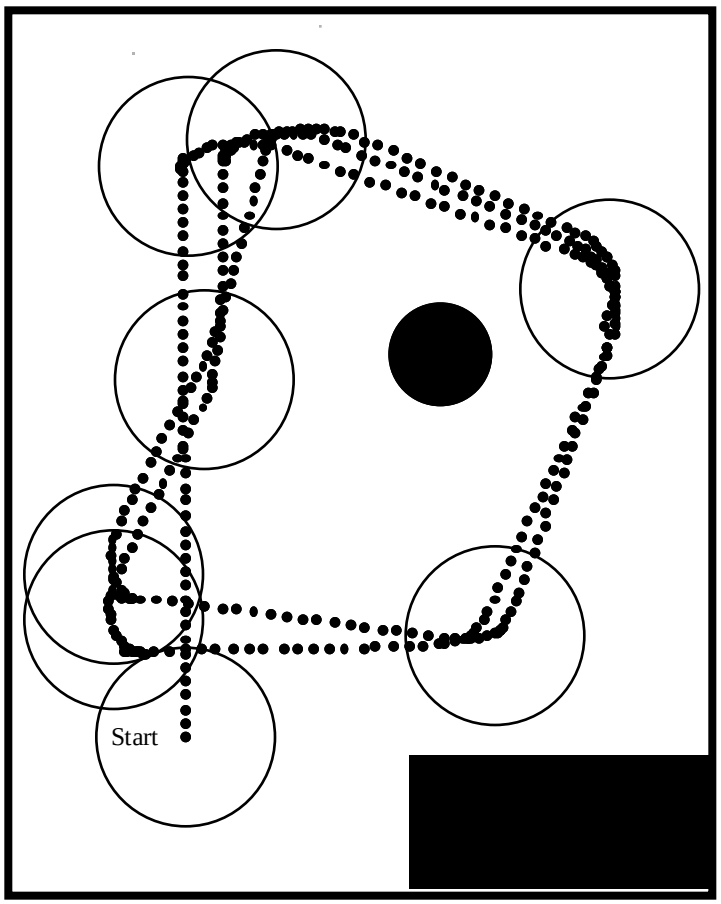


Figure 5. Trace of the Braitenberg avoidance behaviour of Khepera. There are 210 iterations (or moves). The small black circles indicate the robot centre position. The large

circles represent the robot size (5.5 cm in diameter). Two obstacles have been put in the arena (in dark): a diskette box (3.5") and a film can.

Indicator of the task

The distance to the obstacles (G(*t*)) measures the correspondence of the robot's behaviour with the target behaviour. For each obstacle encountered by the robot during the experiment only one value is memorised which is the shortest distance.

G(*t*) = DO(*t*) / *t* ; where DO(*t*) is the sum of the shortest distances from the beginning of the experiment (the higher the sensor value, the shorter the distance to the obstacle). The graph is averaged over seven runs, the mean and standard deviation[1] are displayed. This graph will help us evaluate the training procedures proposed in the following material.

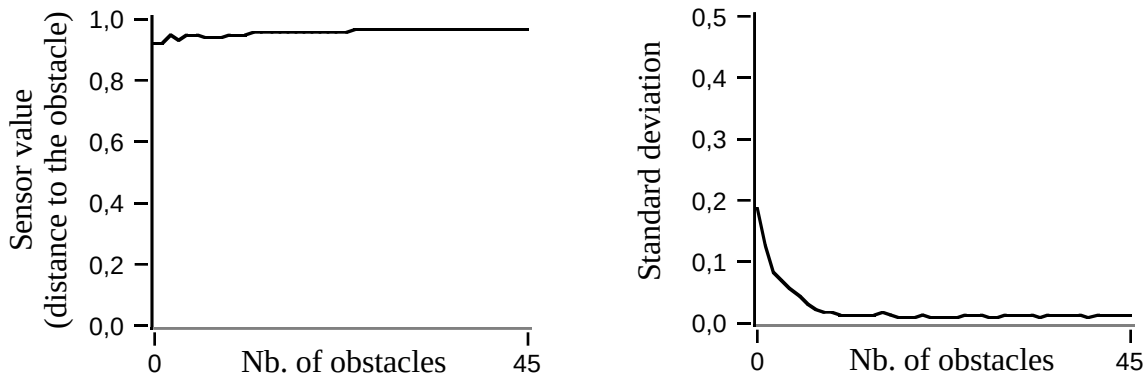


Figure 6. Distance to the obstacles measured during a Braitenberg avoidance behaviour. The results reported are averaged over 7 runs. The robot comes really near to the obstacles (sensor value approximately equal to 1.0).

1.2 Reinforcement function

The robot receives the following reinforcement signals during the learning phase:

+1 if it is avoiding, or
-1 if a collision occurs, or
0 otherwise.

[1] The standard deviation is $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$ (N)

The robot is avoiding when the present sum of sensor values is smaller than the last one, the difference been greater than 0.10. A collision occurs when the sum of the six front sensor values is greater than 2.90 or the sum of the two back sensor values is greater than 1.95. Threshold values like (0.10, 2.90, 1.95) have been determined after extensive experiments. In front of a wall (e.g., the boundary of the arena) a collision corresponds to a distance less than 2 cm. This distance value of 2 cm. is inferred from the sensor performance (fig. 2). It does not necessary mean that the robot touched the wall.

The same absolute value is used for positive and negative reinforcement signals, despite the fact that a collision may be considered much more important than an avoidance action. In the literature, many authors attach more importance to negative rewards, sometimes as much as 100 times more effect on the learning rule. However, we consider the reinforcement function as a qualitative criterion and we do not want to introduce qualitative bias through weighting of the rewards (bias are nevertheless unavoidable: a bias is introduce by any choice of the threshold values).

2. Q-LEARNING

2.1 Reinforcement Learning

Reinforcement learning is the learning of a mapping from situations to actions so as to maximise a scalar reward or reinforcement signal [4]. A robot learns a given behaviour by being told how well or how badly it is performing as it acts in each given situation. As feedback, it receives a single information item from the world. By successive trials and/or errors, the robot determines a mapping function which is adapted through the learning phase. For this purpose, numerous reinforcement learning algorithms are available (e.g., TD(λ), AHC [5]), among which the Q-learning method is certainly the most used.

2.2 Q-learning

Reinforcement learning synthesises a mapping function between situations and actions by maximising a reinforcement signal. Q-learning [6] algorithms store the expected reinforcement value associated with each situation-action pair, usually in a look-up table. Three different functions (figure 7) are involved: memorisation, exploration and updating [7]. Figure 8 describes the Q-learning algorithm.

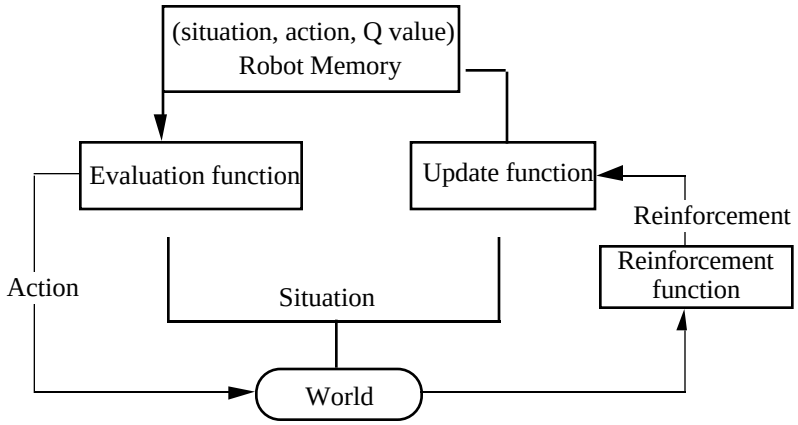


Figure 7. Q-learning method functional decomposition. In response to the present situation, an action is proposed by the robot memory. This action is the one that has the best probability of reward. However, this proposition may be modified by the evaluation function to allow an extensive exploration of the situation-action space. After the execution of the action by the robot in the real world, a reinforcement function provides a reinforcement value. This value, here a simple qualitative criterion (+1, -1 or 0), is used by the updating algorithm to adjust the reward value (*Q*) associated with the situation-action pair. The learning is incremental, because the acquisition of the examples is carried out in real situations.

1. Initialisation of the robot memory: for all situation-action pairs, the associated Q value is 0 (i.e., $Q(i, a) = 0$).

2. Repeat :

a - Let i be a world situation.

b - The evaluation function select the action a to performed:

$a = \text{argMax}_{(a)} (Q(i, a'))$

where a' represent any possible action in situation i .

The exploration process modify the selected action to explore the situation-action space:

$a^* = a + \Delta a$

Δa is usually a randomly selected with a Gaussian distribution $N(0, \sigma)$, σ usually decreases as the learning proceed.

c - The robot executes the action a in the world. Let r be the reward (r can be null) associated with the execution of the action a in the world.

d - Update the robot memory:

$Q_{t+1}(i, a) = Q_t(i, a) + ß(r + \gamma . \text{Max}_{(a')} (Q_t(i', a')) - Q_t(i, a))$. eq. 1

where i' is the new situation after having carried out the action a in situation i ,

a'' is any action which is possible from state i' and $0 < ß, \gamma < 1$.

Figure 8. A general algorithm for the Q-learning method.

A look-up table implementation of Q-learning was used with the miniature robot Khepera to generate an obstacle avoidance behaviour. In all our experiments, ß = 0.9 and $\gamma = 0.5$. For practical reasons, each sensor value is coded as 1 bit (the threshold is 0.2). If the measured value is below the threshold then the sensor bit value is 0, otherwise it is 1. Therefore, the total number of possible situations is restricted to 2^8 (i.e., 256). For the same practical reasons, the total number of actions is reduced to 5 different speeds per motor, reducing the total number of possible actions to 25. Figure 9 gives the size of the look-up table for all the possible coding.

	1 bit	2 bits	3 bits	4 bits	5 bits	6 bits	7 bits	8 bits	9 bits	10 bits
Size	6400	$1.6\ 10^6$	$4.3\ 10^8$	$1.1\ 10^{11}$	$2.7\ 10^{13}$	$7.0\ 10^{15}$	$1.8\ 10^{18}$	$4.6\ 10^{20}$	$1.2\ 10^{23}$	$3.0\ 10^{25}$

Figure 9. Size of the look-up table for all possible coding. There are 25 possible actions per situation and 8 sensors.

The exploration function does not use the classical Gaussian distribution, but a uniform distribution (centred at zero and decreasing proportionately with the number of

iterations). The results of the experiments are presented in two ways: an indicator of the correspondence of the robot's behaviour with the target behaviour, i.e., the distance to the obstacles as previously described (section 1.1) (figure 10) and a local performance index L(*t*) (figure 11) [8].

$L^+(t) = R^+(t) / t$ (respectively $L^-(t) = R^-(t) / t$) measures the effectiveness of the learning process, i.e., the correspondence between what is taught and what is learned. *t* is the number of robot moves executed by the avoidance behaviour module from the beginning of the experiment. $R^+(t)$ (resp. $R^-(t)$) is the number of moves that have been positively (resp. negatively) reinforced from the beginning of the experiment. Graphs are averaged over 5 runs.

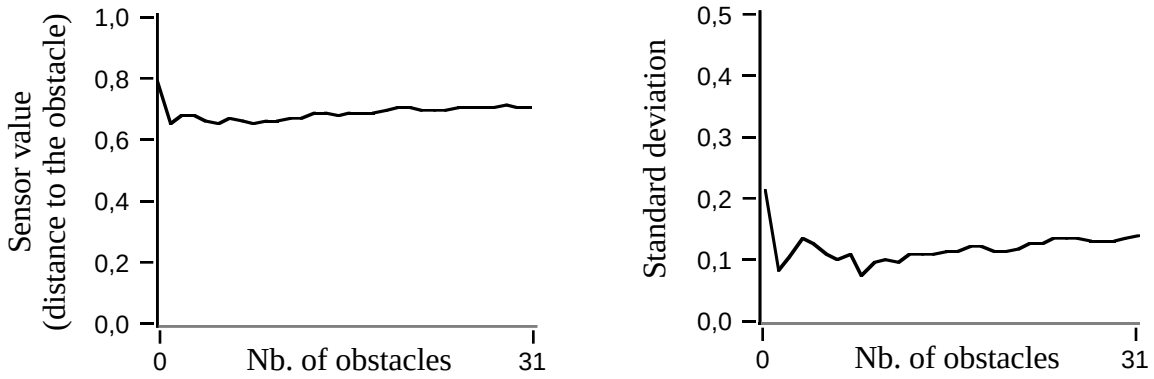


Figure 10. Distance to the obstacles measured after 4000 learning iterations with classical Q-learning. The results reported are averaged over 5 runs, standard deviation is also reported. Distances to obstacles are greater than Braitenberg's implementation (fig. 6): learning is interesting.

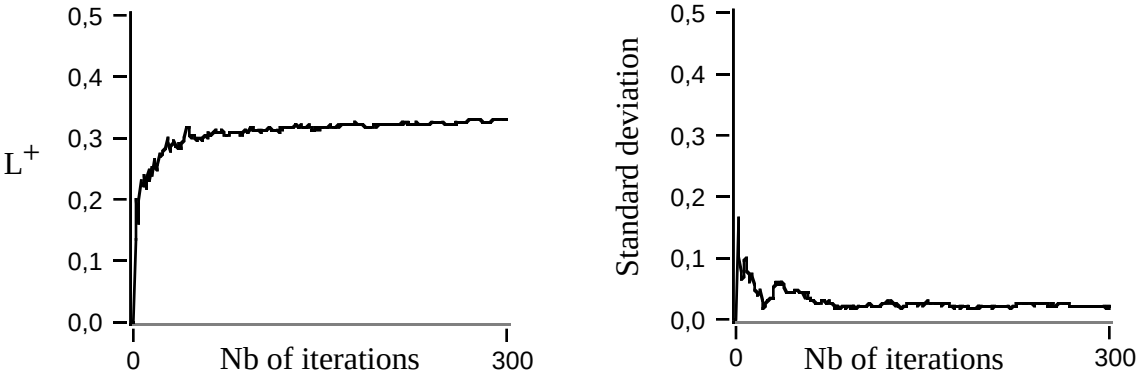


Figure 11. Effectiveness of the learning process (after 4000 learning iterations): proportion of moves executed by the obstacle avoidance module that received positive reinforcements since the beginning of the experiment. Graphs are averaged over 5 runs, standard deviation is also reported.

A trace of the behaviour of the miniature mobile robot Khepera after learning is given in figure 12. The trajectory appears not as smooth as with the Braitenberg vehicle. One of the reasons is certainly the 1 bit coding per sensor (where the Braitenberg vehicle uses 10 bits).

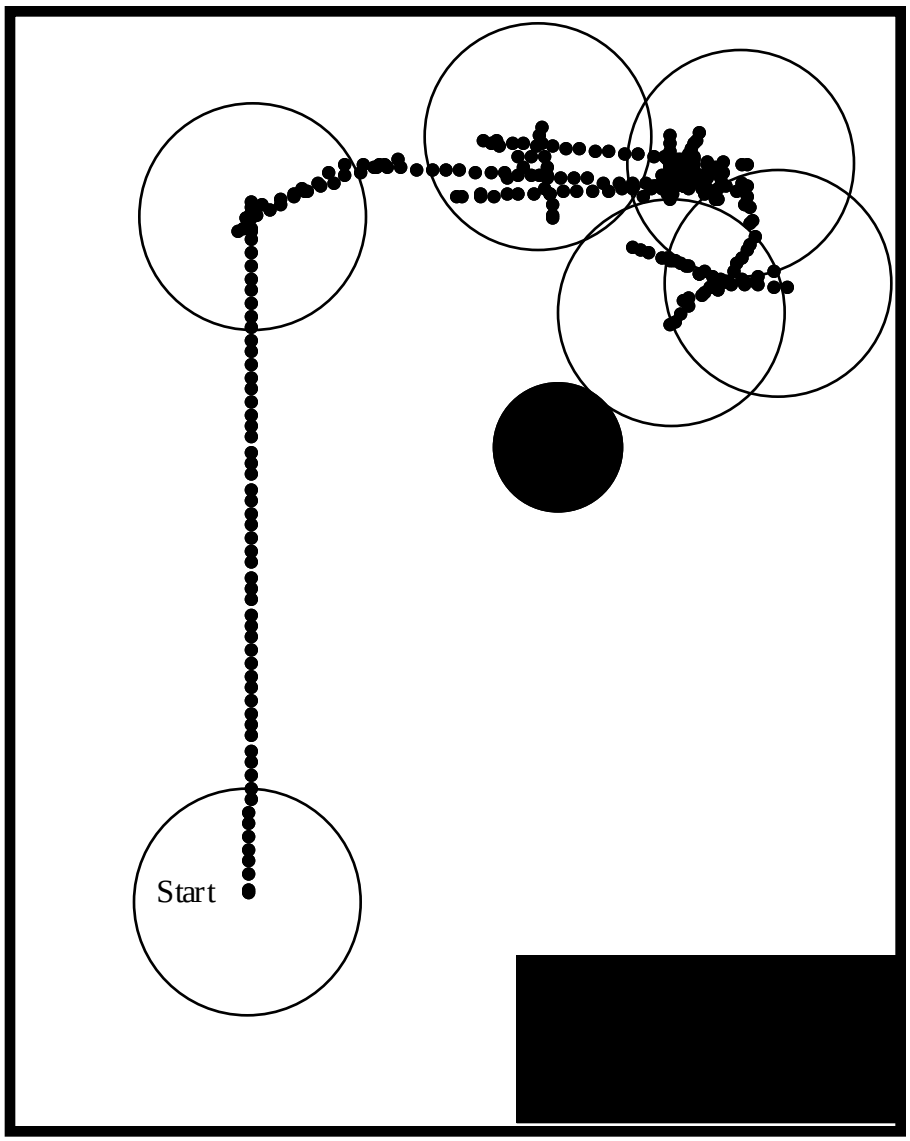


Figure 12. Trace of the Q-learning avoidance behaviour of Khepera after 4000 learning iterations. There are 70 iterations. It is the same environment as fig. 5.

Determining the number of learning iterations required is a difficult task. To this end, we use an indicator: the total number of non-null *Q* values in the look-up table (figure 13). In our case, we choose 4000 learning iterations, i.e., 2/3 of the maximum number of situation-action pairs that can be experienced.

Nb. it.	2 000	3 000	4 000	6 000	8 000	10 000	16 000
%	9.0	10.4	10.8	11.9	12.2	14.6	15.9

Figure 13. Percentages of the total number of non-null *Q* values in the look-up table during a Q-learning synthesis of an obstacle avoidance behaviour. The size of the look-up table is 6400.

It must be reported that, even after 4000 learning iterations, the synthesised behaviour is not perfect. Figure 14 displays the graph of L^- after learning, there are still negative reinforcements experienced, but much less than what a pure random behaviour displays. Moreover, the Braitenberg implementation also experiences negative reinforcements.

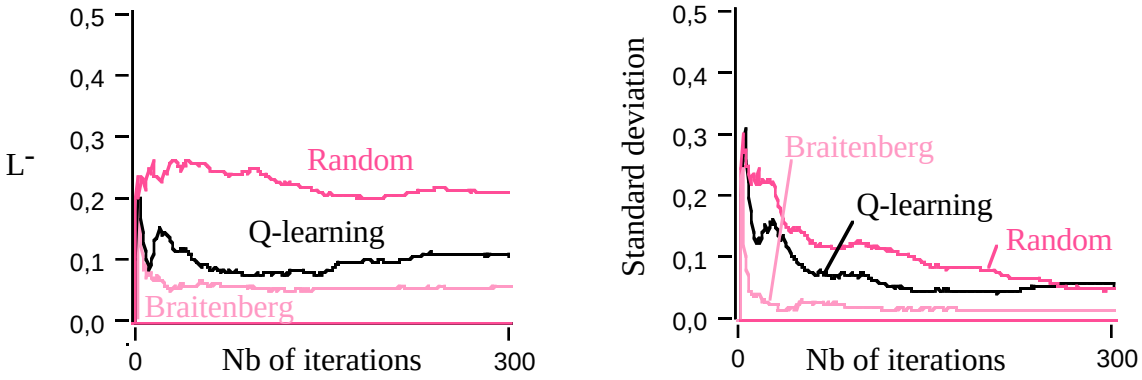


Figure 14. Negative reinforcements experienced with a classical Q-learning implementation after 4000 learning iterations, a random exploration behaviour and a Braitenberg implementation. Graphs are averaged over 5 runs, standard deviation is also reported.

3. CREDIT-ASSIGNMENT PROBLEM

Any difficulty in the use of Q-learning is the result of a situation space being so large that, combined with all possible actions, an exhaustive exploration of all situation-action pairs is impossible, as is also an exhaustive memorisation. For example, in the case of the mobile robot Khepera, the total number of possible situations is $((2)^{10})^8$, not far from 10^{24}. Even if the number of possible speeds per wheel is reduced to 25, the size of the situation-action space is $3\ 10^{25}$. Working with a real robot implies mechanical constraints: in this case, an action takes approximately 200 ms to be performed. In one

minute, a maximum of 300 actions can be executed. Therefore, there is an incredibly small number of explored situation-action pairs versus unknown situation-action pairs. This problem is called the structural credit-assignment problem. A solution to this problem is the generalisation process: the use of experienced situation-action pairs to induce ways of dealing with new unknown situations.

Several improvements emphasising generalisation have been proposed. Mahadevan *et al.* uses Hamming distance to generalise between similar situations; the same authors also use clusters to generalise across similar situation-action sets. Sutton proposes the Dyna-Q model to speed up propagation through time of the *Q* values.

3.2 Q-Learning with weighted Hamming distance [9]

The main idea of this refinement is to compute a Hamming distance between the world situation i and similar situations in order to apply the updating function on all of them. Only one action is carried out, but many similar situations are updated using the same reinforcement value. The Hamming distance between any two situations is simply the number of bits that are different between them. Bits can be of different weights (i.e., weighted Hamming distance). Two situations are distinct if the Hamming distance between them is greater than a fixed threshold (figure 15). This generalisation method is limited to syntactic criteria: it is dependent on the coding of the situations.

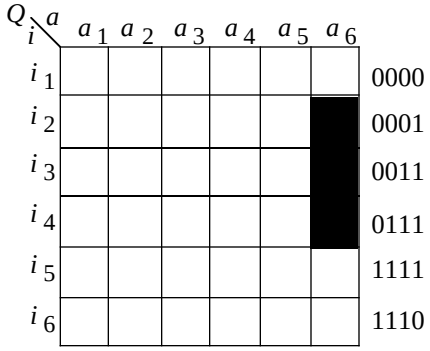


Figure 15. Generalisation using Hamming distance in black. In this example, the world situation is i_3 , the executed action is a_6 , the threshold value for the Hamming distance is 1. Q (i_3 , a_6) is updated, but also Q (i_2 , a_6) and Q (i_4 , a_6).

Experiments with Khepera have been carried out using a Hamming distance of 1 (figures 16 and 17). As one can see, the performance is not high (but worse with a Hamming distance of 2).

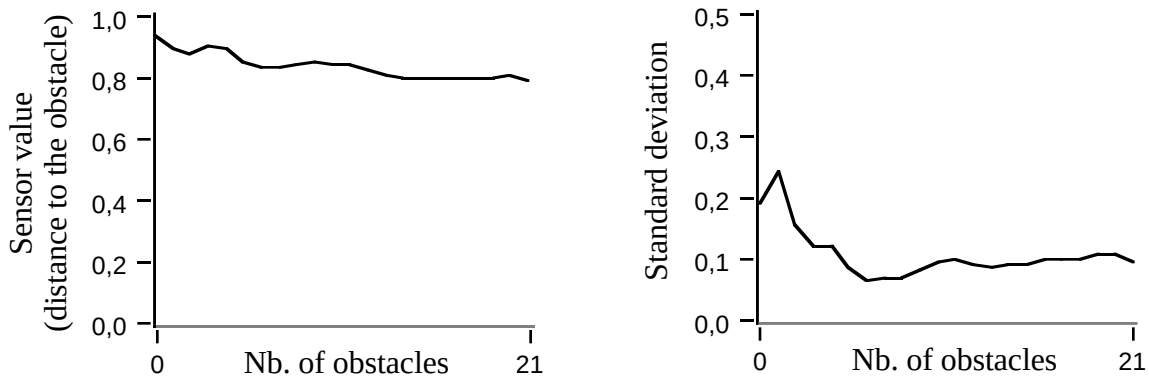


Figure 16. Q-learning with Hamming distance (of 1): distance to the obstacles measured after 2000 learning steps. The results reported are averaged over 6 runs.

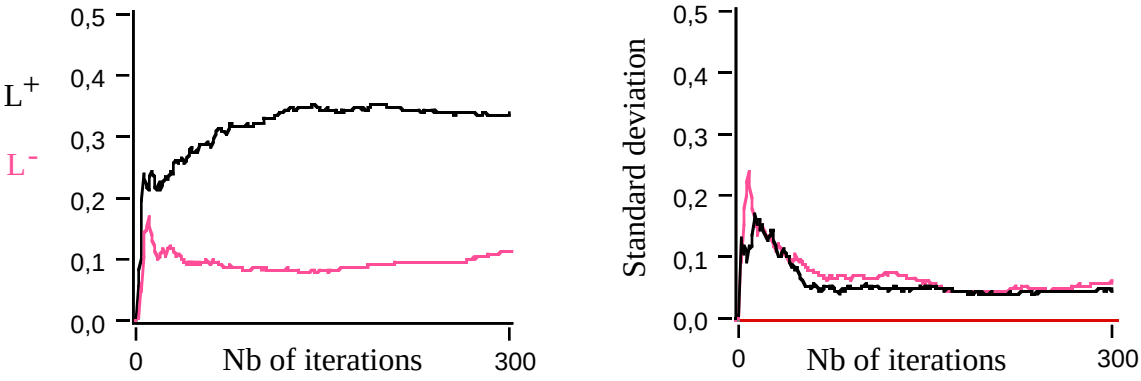


Figure 17. Effectiveness of Q-learning with Hamming distance (after 2000 learning iterations): proportion of moves executed by the obstacle avoidance module that received positive and negative reinforcements since the beginning of the experiment. Graphs are averaged over 6 runs, standard deviation is also reported

The look-up table used is the same as in the previous section. However, learning is sped up through the use of the Hamming generalisation process. Therefore, the total number of learning iterations has been reduced to 2000, i.e., three times (figure 18) the proportion of non-null Q values in the look-up table (fig. 13). It must be pointed out that the generalisation process used here may conduct to invalid actions. For example, when a forwards action is positively rewarded, the generalisation process also rewards the same forwards action in front of small obstacle (only 1 or 2 sensors activated). Therefore, a modification of the Hamming generalisation process has been introduced: statistical clustering

Nb. it.	1000 H1	1500 H1	2000 H1		1000 H2	1500 H2	2000 H2
%	30.8	32.5	32.5		80.8	80.8	82.5

Figure 18. Proportion of non null Q values in the look-up table in function of the number of learning iterations for a Hamming distance of 1 (H1) and 2 (H2). These percentages are between three (H1) and eight (H2) times the maximum value obtained for classical Q-learning. Some of these values conduct to invalid actions.

3.3 Q-Learning with statistical clustering

Mahadevan *et al.* [9] propose an other generalisation method less dependent on the coding of the situations: statistical clustering. Here, each action is associated with a set of situations giving information concerning the usefulness of performing the action in a particular class of situations. Clusters are sets of "similar" situation instances that use a given similarity metric. All situations that appear in the same cluster are updated together (figure 19). Here again, generalisation is limited to syntactic criteria.

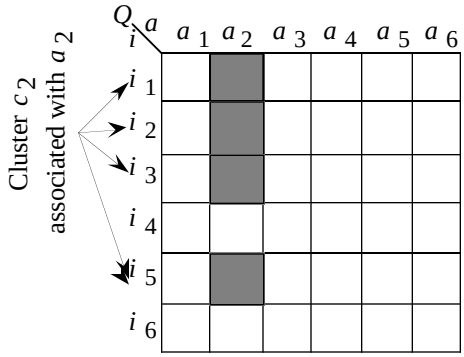


Figure 19. Generalisation using statistical clustering. If one of the situations of the cluster c_2 is the world situation and a_2 as been carried out then all Q values of the cluster c_2 are updated. In this example, the world situation is i_3 and the action carried out is a_2, then $Q(a_2, i_3)$ is updated together with $Q(a_2, i_1)$, $Q(a_2, i_2)$ and $Q(a_2, i_5)$.

Our implementation of the statistical clustering generalisation process on the learning of an obstacle avoidance for Khepera is the following:

- Each forwards action associated with situations involving less obstacles than the resent situation is a cluster (a Hamming distance of 1 is used),
- Each backwards action associated with situations with no obstacles behind is a cluster (a Hamming distance of 1 is used),
- It is not legitimate to have cluster associated with rotational actions.

Results are displayed in figures 20 and 21. It seems that clustering is not better, even worse, than the Hamming generalisation in our case. The proportion of non-null Q values is 17.1% after 2000 learning iterations.

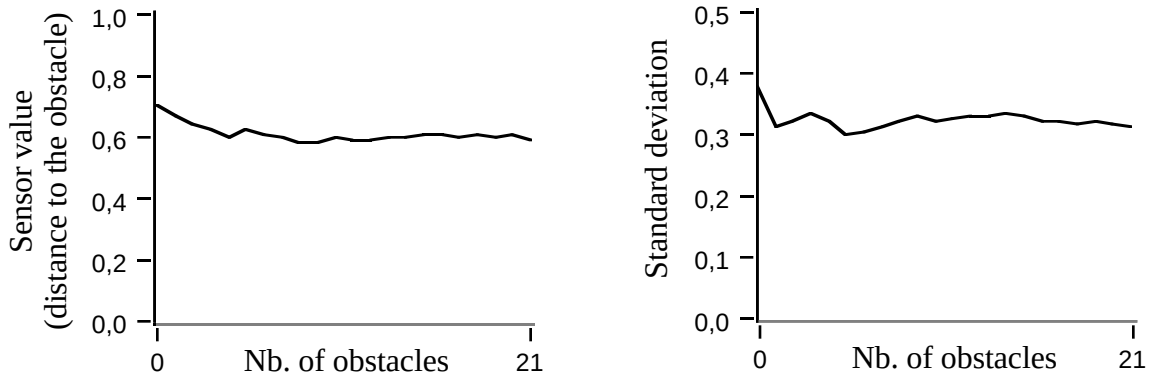


Figure 20. Q-learning with statistical clustering : distance to the obstacles measured after 2000 learning steps. The results reported are averaged over 5 runs. As shown by the standard deviation graph, the synthesised behaviours are very different in their performances.

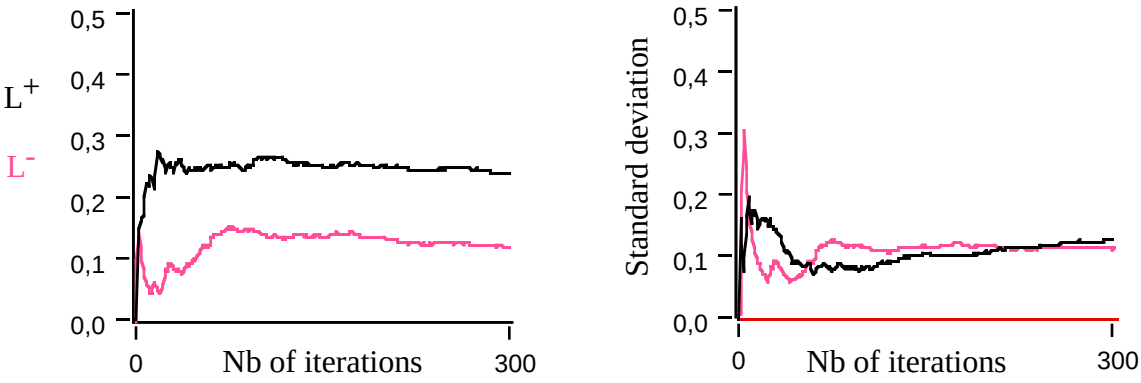


Figure 21. Effectiveness of Q-learning with statistical clustering (after 2000 learning iterations): proportion of moves executed by the obstacle avoidance module that received positive and negative reinforcements since the beginning of the experiment. Graphs are averaged over 5 runs, standard deviation is also reported.

3.4 Dyna-Q

It may be difficult to rerun the same sequence of situation-action pairs so as to back propagate delayed rewards (equation 1 in figure 8). As a result, Sutton [10] imagined

adding a model of the world in which situation-action pairs are randomly carried out again (figure 22). The reinforcement function is modified in order to deal with the modelled world. In this case, only situation-action pairs previously seen (in the real world) will lead to a non-zero reinforcement reward. The returned reward *r'* is the same as in the real world. When the experience is performed in the real world, the exploration process is the same as described earlier in section 2.2. Otherwise (i.e., for an experience in the modelled world) the exploration function is a random selection of actions.

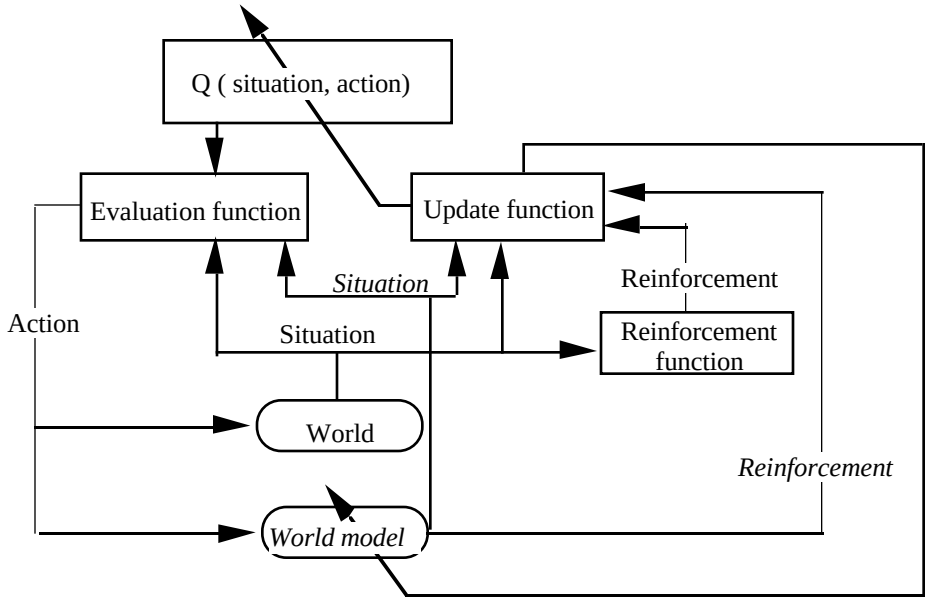


Figure 22. Dyna architecture: the world model is used as a direct replacement for the world. In addition to the usual functioning, learning steps are also run using the model in place of the world, using predicted outcomes rather than actual ones. For each real experience with the world, many hypothetical experiences (randomly) generated by the world model can also be processed and learned from. The cumulative effect of these hypothetical experiences is that the policy approaches the optimal policy given the current model.

The same look-up table is used as in the classical Q-learning implementation. This implementation needs fewer actions carried out in the real world, but the number of learning iterations is increased. The proportion of non-null *Q*values in the look-up table is 19.4% after 1000 learning iterations and 26.7% after 2000 learning iterations. There are ten hypothetical experiences for each real world action (the maximum possible number is 25: the number of possible actions). The ten hypothetical actions are randomly selected using a uniform probability. The results obtained with Khepera for an obstacle avoidance behaviour synthesis are given in figures 23 and 24.

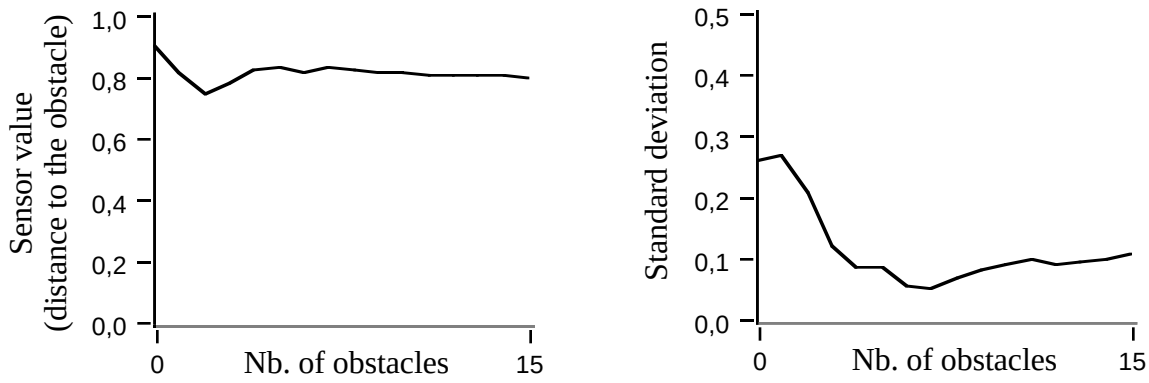


Figure 23. DYNA Q-learning: distance to the obstacles measured after 1000 learning iterations. The results reported are averaged over 5 runs.

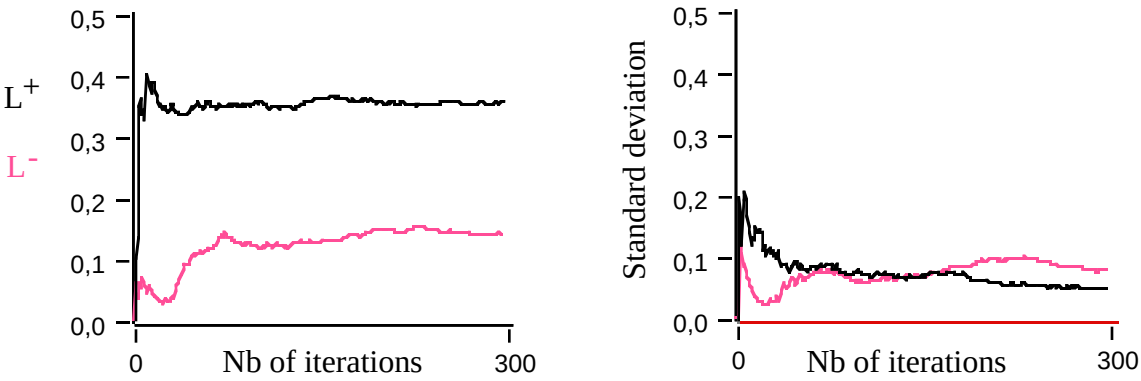


Figure 24. Effectiveness of DYNA Q-learning after 1000 learning iterations : proportion of moves executed by the obstacle avoidance module that received positive and negative reinforcements since the beginning of the experiment. Graphs are averaged over 5 runs, standard deviation is also reported.

3.5 Comparisons

Experiments on the learning of an obstacle avoidance behaviour for the mobile robot Khepera show that these three methods speed up the learning (figure 25). Our criterion for having learnt is no more collision occur (estimated subjectively by the experimenter), together with a minimum number of positive rewards (e.g., the behaviour that consists in stopping far from any obstacle is good at experiencing no collision, but not with reference to the minimum of positive rewards). The minimum positive rewards requested is chosen by the experimenter.

The performances after learning are not perfect. This comes from the fact that a unique bit is used to code each sensor value (with a threshold at 0.2). This is certainly not precise enough. However, it is impossible to use all the bits available per sensor (10). The generalisation is limited to syntactic criterion (Hamming and clustering). There is no specific generalisation with the Dyna-Q implementation, just help in the propagation of the reward signals.

AHC [5] is very similar to Q-learning. The major difference is that AHC builds two functions (one for evaluating the interest of the situation, one for proposing an action in a situation) where Q-learning builds only one function (evaluation and policy in the same time). We think that the results reported here should also apply to the AHC method, but the necessary experiments were not conducted.

	Learning time (sec)	Nb. of learning it.	Memory size (%)
Q-learning	1680	4000	10.8
+ Hamming (1)	1020	12000	32.5
+ Clustering	860	6000	17.1
DYNA-Q	540	11000	19.4

Figure 25. Comparison of several different implementations of Q-learning in terms of learning time, number of learning iterations and memory size. The learning time is the time in seconds needed to synthesise an obstacle avoidance behaviour. It reflects the number of real world experiments required. The number of learning iterations is the number of updates of the look-up table (after real world or modelled world experiments). The memory size is the proportion of non-null Qvalues in the look-up table (size 6400).

4. NEURAL Q-LEARNING

The first neural network implementation of reinforcement learning occurred at the beginning of the eighties [11]. Neural Q-learning implementations were proposed in the nineties[12]. Neural implementation seems to offer many advantages: quality of the generalisation and limited memory requirement for storing the knowledge. The memorisation function uses the weight set of the neural network. The memory size required by the system to store the knowledge is defined, a priori, by the number of connections in the network. It is independent of the number of explored situation-action pairs.

4.1 Ideal implementation

The ideal neural implementation will provide, in a given situation, the best action to undertake and its associated *Q* value (figure 26). This action should be the best available action in the situation, but how can we be sure of that?

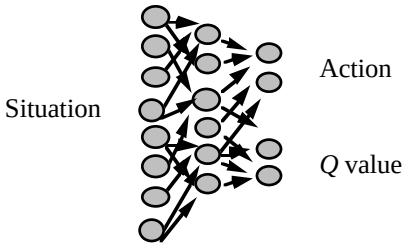


Figure 26. Ideal neural Q-learning model providing for an input situation the best action to undertake and its associated *Q* value.

The direct implementation of the ideal neural network implementation of Q-learning with a multilayer backpropagation network implicates several drawbacks. The updating function (figure 27) is a weight modification algorithm, here the well-known gradient error backpropagation algorithm [13]. An error signal on the output neurones must therefore be defined for each output neurone. How can a quantitative error signal be defined when the only available information is of qualitative nature? The definition of this error is restricted to simple cases where only two actions are possible. For example, one of the first applications used to demonstrate the power of the neural implementation of Q-learning was the inverse pendulum with only two actions (moving the cart left or right). In this case, it is easy to deduce from the reinforcement signal the desired output value. If the reinforcement signal received is positive (+1) then the output provided by the neural network is the desired one. If the reinforcement signal received is negative (-1) then the desired output is the inverse of the output proposed by the network. For applications involving many possible actions, like mobile robotics (e.g., Khepera allows 400 different actions), dealing with negative rewards is more difficult and requires modifications of the ideal implementation.

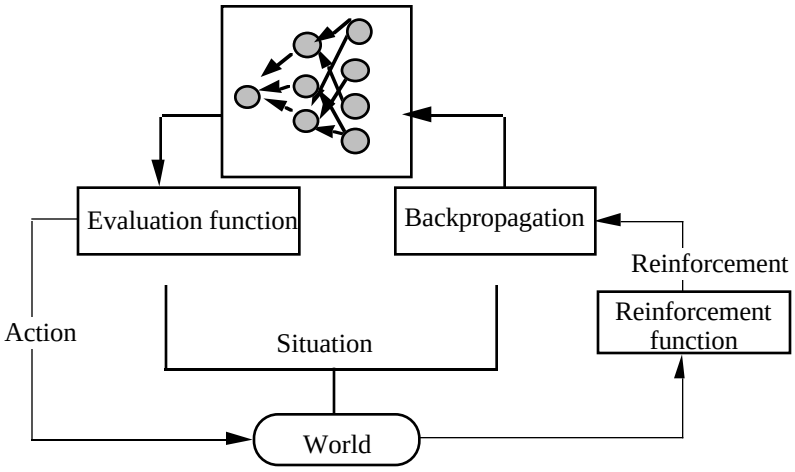


Figure 27. The direct implementation of the ideal neural network implementation of Q-learning with a multilayer backpropagation network.

4.2 QCON

Lin [14] proposes the QCON model: a multilayer perceptron implementation of the Q-learning algorithm the characteristic of which is to have only one output neurone. There are as many QCON networks as there are actions (figure 28). It is impossible to generalise across actions.

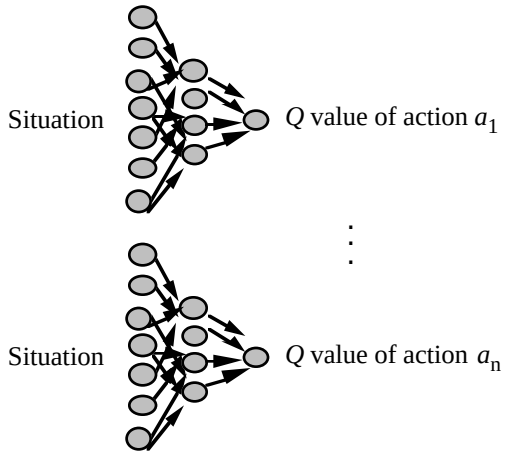


Figure 28. The QCON model: only one output neurone per multilayer neural network. Each network is associated with a unique action and the output is considered as the *Q* value associated with the situation-action pair. There are as many QCON networks as there are actions. Generalisation across situation-action pairs is impossible. Learning concerns only one network per iteration.

4.3 Unmapping in multilayer backpropagation neural networks

Generalising across actions implies an implementation architecture composed of only one network so as to be able to use every reinforcement signal to update the network. Moreover, generalising across actions implies that the output layer codes actions and not *Q* values. Since the neural model is still a multilayer perceptron with a backpropagation learning algorithm, an error must be defined on the output layer. There is no problem when the reinforcement signal is positive, the proposed action is the desired one. But, with negative reinforcements, how can a desired action be chosen? At least, even if it is not possible to find the right situation-action association, a mechanism must be built that will unmap the wrong situation-action association proposed by the network.

The first idea that comes to mind when the number of possible actions increases is to learn the inverse mapping [12]. The problem then is to determine among all those left which action is the most in opposition. Because the output of the network is numerical, we can change the sign of the output values. However, this is a harmful way of unlearning. Nobody knows what has been deleted. Representation on a neural network is distributed, so it is not possible to delete only one association (or situation-action pair) without interfering with the rest of the learned knowledge. Moreover, a negative reinforcement does not always mean that the error is important, and to learn the inverse action can be defective. With the same goal in mind, Ackley [15] proposes the use of the complement of the generated output. Results are completely dependent on the nature of the application and are not satisfactory. We propose in the next section an action decomposition mechanism that allows unmapping without destroying the memory of the network.

4.4 Modifying the reinforcement function

Since problem of unmapping only arises with negative reinforcement signals, it may be an interesting solution to modify the reinforcement function. However, it is impossible to learn, in the general case, using only positive rewards. Know-how, on the contrary, tells us that it is frequently simpler to refine reinforcement functions with negative reinforcements.

5. A COMPETITIVE MULTILAYER PERCEPTRON FOR THE IMPLEMENTATION OF THE Q-LEARNING

As opposed to the implementations seen in section 2 and 3, neural reinforcement restricts learned knowledge. The ideal neural network implementation (section 4) has the great advantages of enormously reducing the memory requirements. However, the executed situation-action pairs are not explicitly stored, nor is the associated *Q* value. Only the most important feature of Q-learning is conserved: to be able to propose the action with the best reward potential for a given world situation. The more realistic QCON proposal does not take full advantage of a neural implementation. As many networks as the number of actions are necessary, greatly limiting generalisation.

We propose a neural implementation model that is intermediate to the ideal implementation model and QCON (it allows us to generalise between actions, but the associated *Q*value is not stored explicitly):

Step 1: Actions are grouped together in sets of agonist-antagonist actions. This can be done through a decomposition mechanism starting with complex actions and ending with the elementary actions as shown in figure 29.

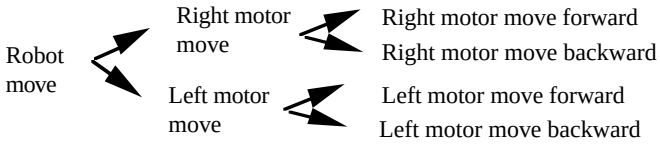


Figure 29. Example of decomposition into elementary agonist-antagonist action sets: from the complex action 'robot move' to the elementary sets of actions, like 'right motor move forward'.

Step 2: These sets of elementary actions are coded on the output layer of the unique multilayer perceptron. One output neurone is used for each elementary action set (for the example given in figure 29, there are four output neurones). Into the set, the action to perform is coded by the output value of the neurone. In the example of the mobile robot Khepera, we use ten speeds per motor. Each neurone output varies from -1 to +1 which correspond respectively to speed 0 and speed 9. We chose a uniform distribution for the other values. It is extremely important for the generalisation process to respect continuity in the coding of the action set (order preserving coding). Therefore, speed 0 must be coded by a value less different from speed 1 than speed 9 (respectively -1 , -0.8 and +1).

Step 3: Having two neurones per motor (one for forward speeds and the other for backward speeds), the system has to decide which actions will be executed. We choose to interpret the neurone output value as a certainty value of the neural network in its proposition. Interpreting the network output values as certainty values is a common practice in connectionism, in particular in the domain of fuzzy reasoning [16]. A competition between the two 'agonist-antagonist' neurones allows us to select the actions to undertake (figure 30). It is important to note that the fastest speed is always selected, either forwards or backwards. For example, it is extremely unlikely that a stop action will be selected. The following assumption is made: a non-null motor speed always allows us to avoid the obstacle.

Step 4: If the reinforcement signal received after carrying out the action is positive, then the desired output is the executed action (figure 30) and backpropagation can occur.

Step 5: If the reinforcement signal received after carrying the action is negative, then the value of each agonist-antagonist pair is exchanged. The desired output is the counterpart value of the executed action (figure 30) and backpropagation can occur. This method allows us to limit the size of the error, which depends on the certainty of the network.

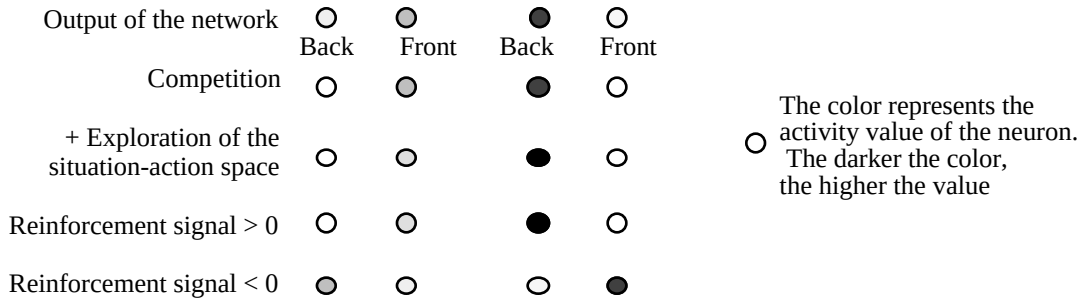


Figure 30. Through competition between the neurones in each pair, the neurone of maximum value is selected. The exploration process can modify the proposal so to explore the situation-action space. If the reinforcement signal is positive then the error is equal to the value added by the exploration process. There is no error for the other neurone of the pair. If the reinforcement signal is negative then values of each pair of neurones are exchanged.

The five steps of this method for a competitive multilayer neural network implementation of Q-learning are summarised in figure 31.

1. Random initialisation of the network weights.
2. Repeat :
   a. Let i be an input situation for the neural network, and (o_1, o_2) the outputs computed by propagation of activities. The action a to perform is given by competition :

   $a = o_{winner} + \Delta o$,

   where Δo comes from the exploratory process as described in 2.2 and

   o_{winner} = Max. (o_1, o_2).

   b. Execute the action a in the world.

   c. Let r be the reinforcement signal associated with the execution of a in i . The weights are updated by an algorithm which minimises the output error. It is necessary to determine a desired output value d for each output neurone, depending on r .

   If r = 0 then there is no modification.

   If r = +1 then $d_{select} = a$. Only weights connected to the selected neurone are modified.

   If r = -1 then $d_1 = o_2$ and $d_2 = o_1$ (exchange of values). All weights are modified.

Figure 31. Learning algorithm for a competitive multilayer neural network implementation of the Q-learning.

Experiments with the competitive multilayer perceptron implementation of Q-learning on the synthesis of an obstacle avoidance behaviour for the miniature mobile robot Khepera are displayed in figures 32 and 33. The multilayer perceptron has only one hidden layer of 4 neurones, an output layer of 4 neurones and 8 input neurones. All input neurones are connected to the hidden and output neurones. There is a threshold connection on each hidden neurone. The total number of connections is 84.

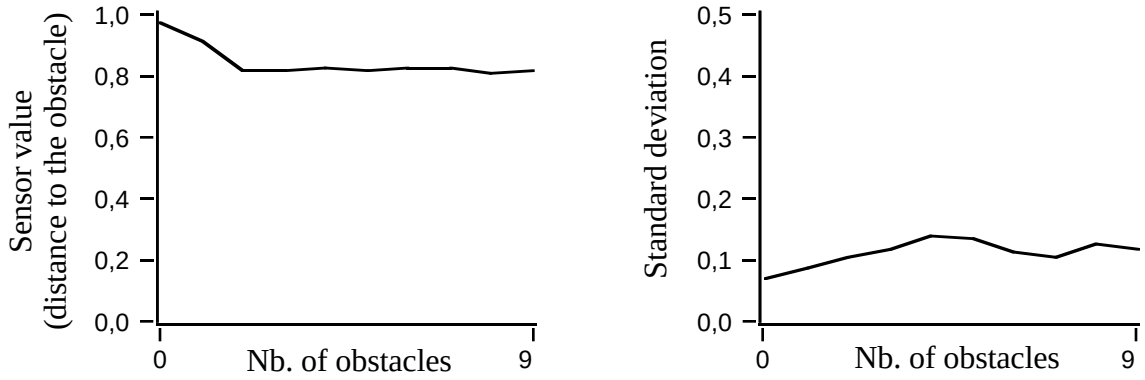


Figure 32. Competitive multilayer neural network implementation of Q-learning: distance to the obstacles measured after 500 learning iterations. The results reported are averaged over 5 runs.

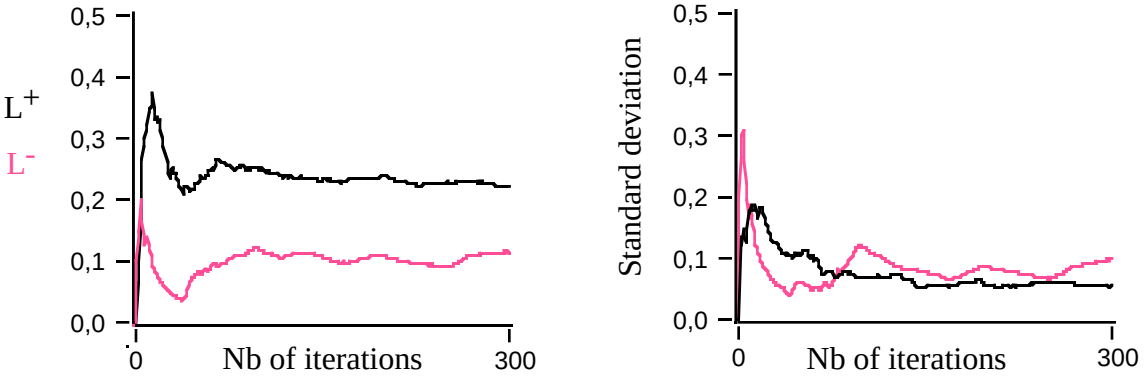


Figure 33. Effectiveness of Q-learning with a competitive multilayer neural network implementation (after 500 learning iterations): proportion of moves executed by the obstacle avoidance module that received positive and negative reinforcements since the beginning of the experiment. Graphs are averaged over 5 runs, standard deviation is also reported.

6. SELF-ORGANISING MAP Q-LEARNING

Experiments reported in the previous section show that the competitive multilayer perceptron implementation learns faster than the other reviewed implementations: generalisation is better. We think that this is due to the localised coding on the output layer: one output neurone for each action set; and also to the competition between output neurones. Therefore, we chose to investigate the implementation of Q-learning with a neural network model that is completely dedicated to these two points: the self-organising

map (also known as the Kohonen map) [17]. We call this self-organising map implementation of the Q-learning Q-KOHON. There are other competitive neural network models which have the same properties: ART [18], CMAC [19], etc.

6.1 Self-organising map

Structure

There is only one layer of neurones. Each neurone is connected to the inputs. A neighbourhood is defined for each neurone. The neighbourhood size depends on the application. In this paper, we use four neighbours per neurone.

Learning

Coding on a self-organising map is localised. Each neurone represents a particular class (or cluster) of the inputs (figure 34). Competition occurs between all the neurones of the map. During the learning phase, the neurones of the self-organising map approximate the probability density function of the inputs. The learning algorithm is presented in figure 35.

Repeat

For each example E of the learning base, E = {x,y}

For each neurone i of the SOM,

$d(i)= |x - w_{x,i}| + |y - w_{y,i}|$ (equation 2)

Select the closest neurone: j = argMin (i) (d(i))

Modify the weights of j using a learning coefficient $0<\lambda<1$:

$w_{j,x}(t+1) = w_{j,x}(t) + \lambda.(x - w_{j,x}(t))$,

$w_{j,y}(t+1) = w_{j,y}(t) + \lambda.(y - w_{j,y}(t))$

Modify the weights of the neighbours of j with a learning coefficient μ using the same rule (usually $0<\mu<\lambda$)

Figure 34. Self-organising map learning algorithm.

Learning projects the N dimensional space represented by the training data on the M dimensional space of the self-organising map (usually two dimensions). This projection respects the relative distribution of the training data. The neighbourhoods allow us to generate a "continuous" mapping of the N dimensions space on the self-organising map. Close N dimensional data are close in the M dimensions of the self-organising map.

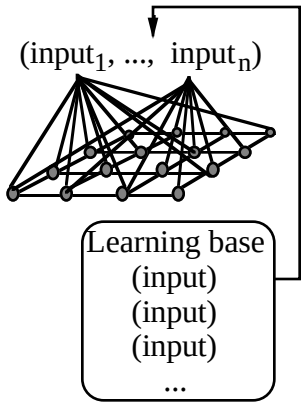


Figure 35 Self-organising map model. During the learning phase, the network weights are tuned so as to reduce the difference between the input to the network and the selected neurone.

6.2 Q-KOHON

During the learning phase, the neurones of the self-organising map approximate the probability density function of the inputs. The inputs are situation, action and the associated Q value (figure 36). The learning phase associates with each neurone of the map a situation-action pair plus its Q-value. It is a method of state grouping involving syntactic similarity and locality [20]. The number of neurones equals the number of stored associations. The neighbourhood property of the Kohonen map allows it to generalise across similar situation-action pairs.

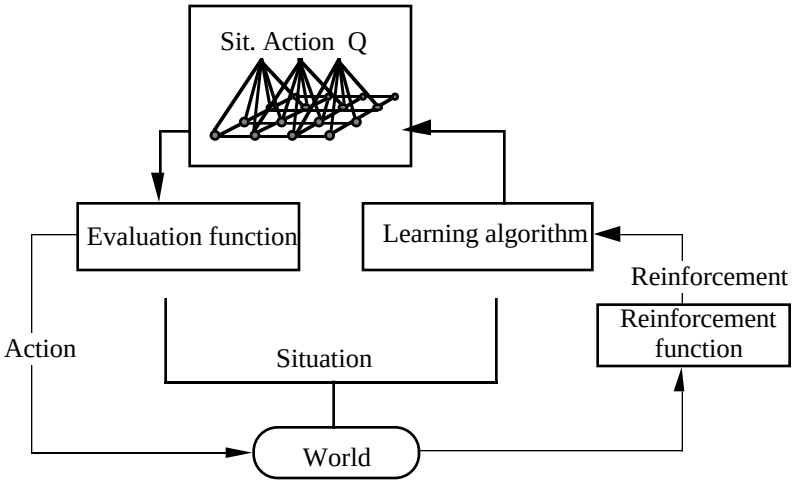
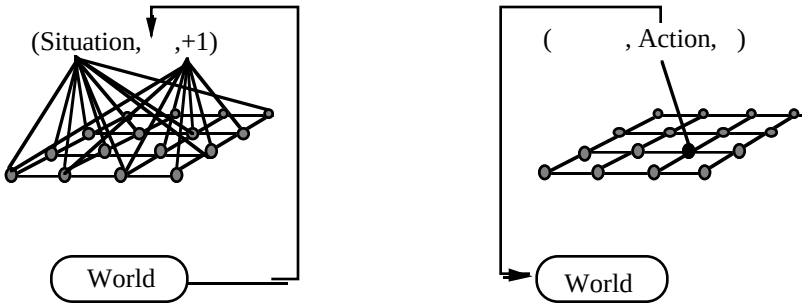


Figure 36. The self-organising map implementation of Q-learning.

The Q-KOHON uses the self-organising map as an associative memory. This associative memory stored triplets. Part of a triplet is used to probe the self-organising map in search of the corresponding information. Here, situation and *Q*value are used to find the action: the best action to undertake in a world situation is given by the neurone that has the minimal distance to the input situation and to a *Q* value of value +1 (figure 37 a). To this end, equation 2 in the self-organising map learning algorithm (figure 34) has been changed to:

$$d(i)= |world\_situation - W_{situation,\ i}| + |1 - W_{Qvalue,i}|$$

The selected neurone corresponds to a triplet (situation, action, *Q* value). It is this particular action that should offer the best reward in the world situation (figure 37 b).



a b

Figure 37. Selection of the best action to perform in the world situation. The Kohonen map is used as an associative memory: information is probed with part of it.
a/ The world situation and a *Q* value of +1 are given as inputs.
b/ The answer is a selected neurone which weights give situation, *Q* value and the associated action.

Training:

The learning base is built incrementally (as for the competitive multilayer implementation). Each action of the Khepera robot is a learning example: the number of learning iterations is the total number of experiments (actions carried out in real world situations).

The learning algorithm updates the *Q* value weight (using equation 1) and, also, the situation and action weights. The neurone corresponding to the situation and the action effectively performed is selected. The distance used is different from the search process of the most rewarding action. It includes the situation and action vectors, but nothing concerning the *Q* value. The equation 2 is modified in the following manner:

$$d(i)= |world\_situation - W_{situation,i}| + |1 - W_{action,i}|$$

Together with the selected neurone, the four neighbours are also updated. The learning coefficient is 0.9 for the selected neurone and 0.5 for the neighbourhood. During the

learning, the influence on the neighbours decreases proportionally to the inverse of the number of iterations. Results are given in figures 38 and 39. 100 iterations are sufficient to learn a correct behaviour. There are sixteen neurones in the map (176 connections (11 x 16)). The weights are randomly initialised around 0.5 for sensors and 0.0 for motors. *Q*values are initialised to 0.0. The inputs are of three kinds: a situation vector, an action vector and a *Q* value. The first eight inputs correspond to the eight sensors. In this neural implementation, the ten coding bits of each sensor are used. The ninth and tenth inputs correspond to the left and right motor commands respectively. The eleventh input corresponds to the *Q* value.

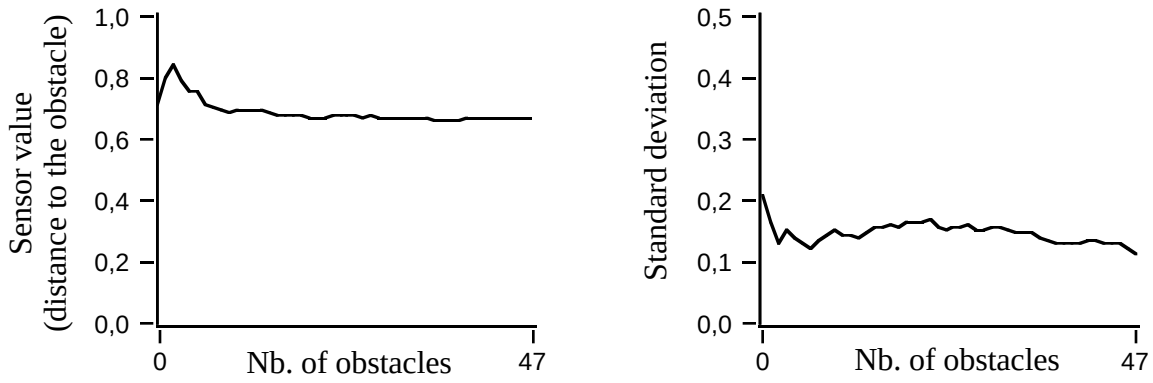


Figure 38. Q-KOHON : distance to the obstacles measured after 100 learning iterations. The results reported are averaged over 5 runs.

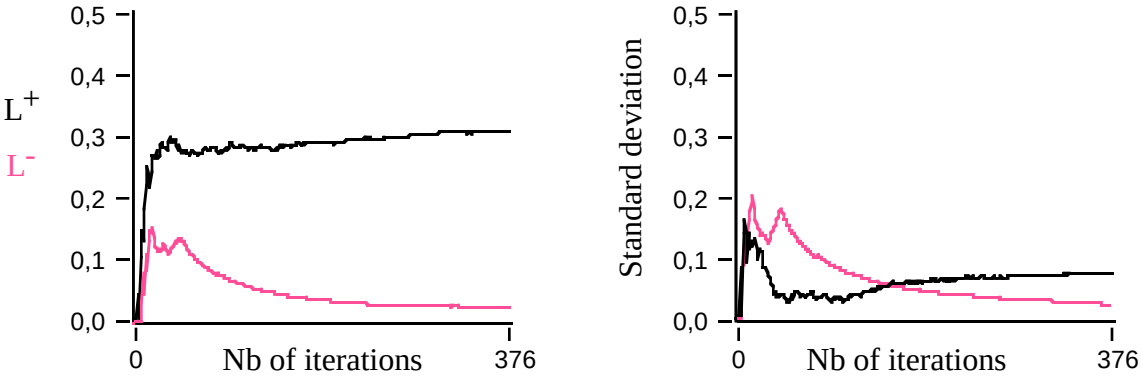


Figure 39. Effectiveness of Q-KOHON (100 learning iterations plus 276 test iterations): proportion of moves executed by the obstacle avoidance module that received positive and negative reinforcements since the beginning of the experiment. Graphs are averaged over 5 runs, standard deviation is also reported.

It is also interesting to visualise the performance traces of the robot after learning (figures 40 and 41)

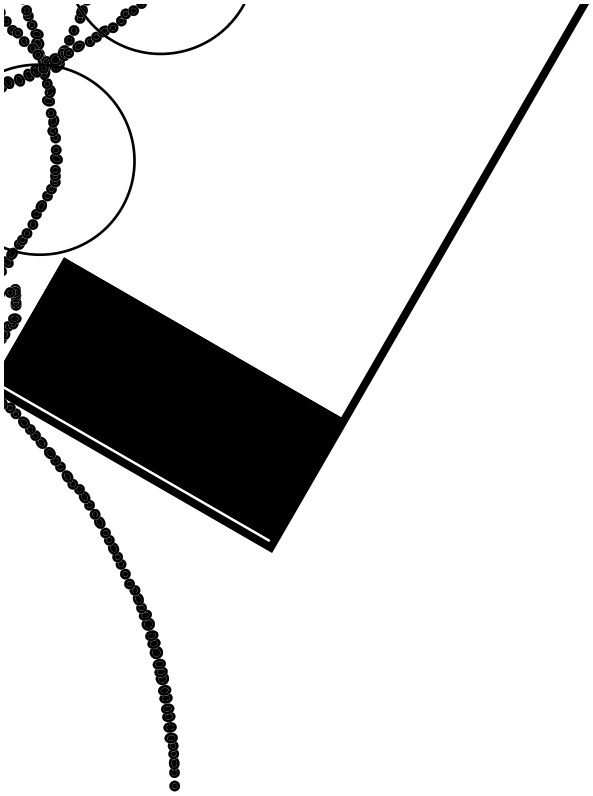


Figure 40. Performance traces (118 iterations) of the robot with Q-KOHON after learning.

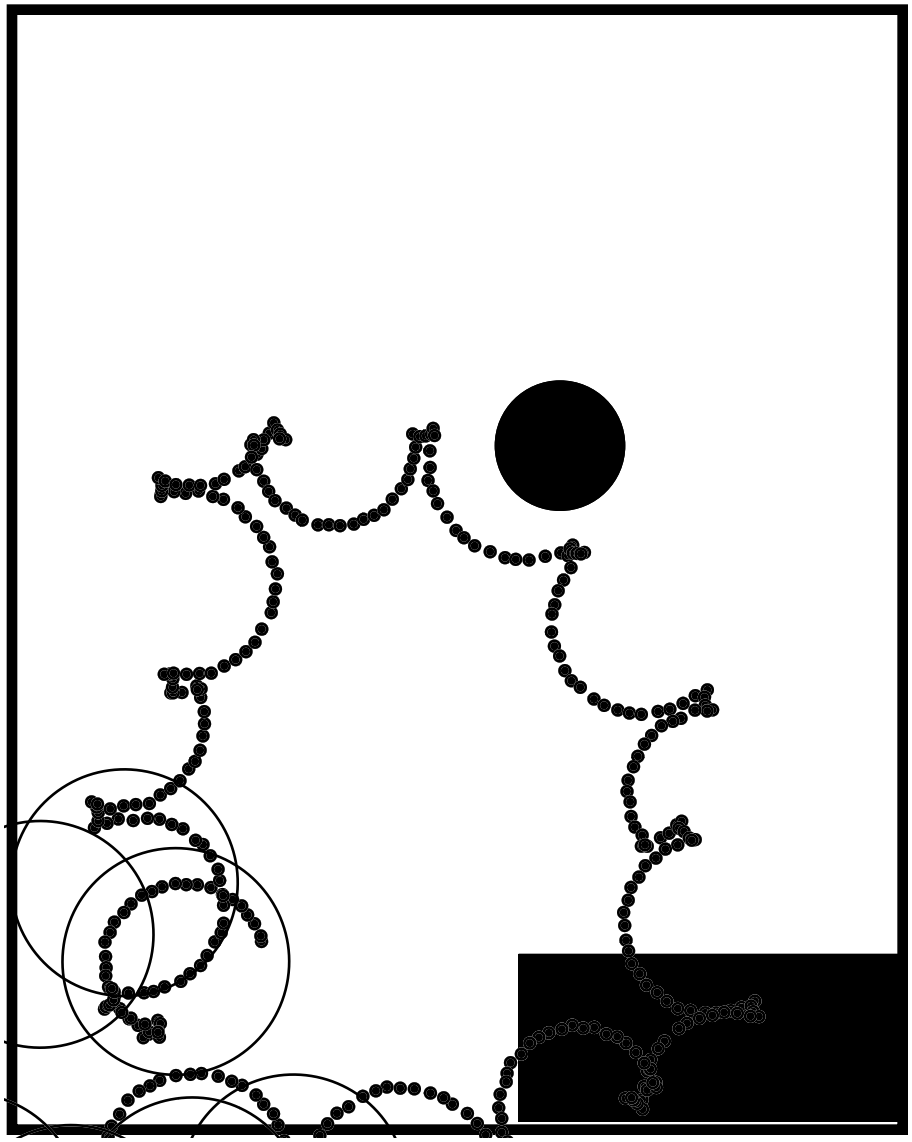


Figure 41. Performance traces (76 iterations) of the robot with Q-KOHON after learning.

Studying the network weights, we can interpret the learned behaviour. Figure 42 presents for each neurone its eight weights linked to the situation input after 100 learning iterations. These weights only depend on the explored situations and represent for each neurone the prototype situation associated. For example the diagram for the neurone number one (N1 above left) shows that N1 is associated with a situation presenting an obstacle on the right side, principally coded by sensor 6, but also to a lesser level by sensors 5 and 7. This obstacle must be very near (about 2 cm), since the value corresponding to sensor 6 is equal to 1. This obstacle is large enough to be detected by sensors 5 and 7, and to a lesser extent by sensors 4 and 8. One should recall the heterogeneity of the sensors (figure 2) when looking at the prototype of each class of situations, particularly evident in N11 with the less effective sensor 5. All kinds of situations encountered during the learning are represented here. Due to the topological constraints of the maps, neighbouring neurones are associated with similar situations;

similar sensors involved (shape of the obstacle) and similar values (distance to the obstacle).

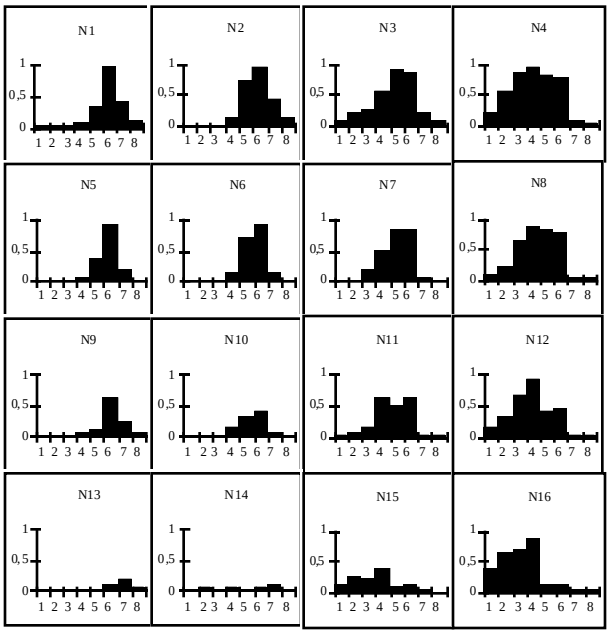


Figure 42. Visualisation of the eight weights linked to the situation input for each neurone of the map after 100 learning iterations in a task of obstacle avoidance behaviour synthesis. The higher the value of the sensor, the more sensitive the corresponding neurone to an obstacle. These diagrams represent the sixteen shapes of obstacles used for the classification. Each class is associated with an appropriate action.

The properties of the self-organising map allow us to predict that, if a correct behaviour is learned (i.e., only positive rewards are experienced), then all neurones will code positive *Q* values. Figure 43 presents the evolution of the weights corresponding to the *Q* value during the learning phase for all neurones. For N2, the curve shows that the *Q* value of the associated situation-action pair starts initially at 0.0, decreases to -0.5 and after 100 iterations, increases to 0.65. Initialisation puts all values to 0.0. The *Q* values become positive for all neurones: the actions undertaken in the encountered situations are correct. Figure 44 displays the mean of the *Q* values of all the neurones during the learning phase: the global performance of the robot is increasing, i.e., something rewarding is learned.

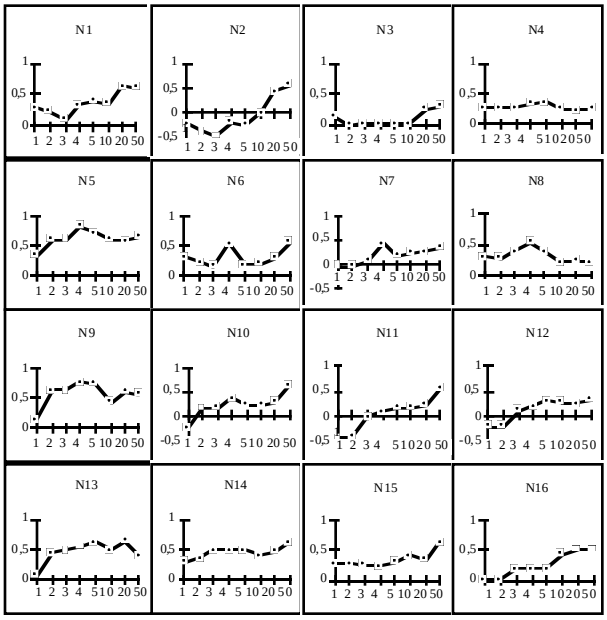


Figure 43. Evolution of the *Q* value associated with each neurone (or class of situation-action represented by a neurone). The *Q* values, starting from 0.0, converge to positive values for all neurones, demonstrating that the learned behaviour is rewarding. There are 100 learning iterations, but the learning does not completely stop after 100 ($\lambda = \mu = 0.001$). After 500 iterations, only situation-action pairs giving positive rewards are stored in the map. The number of iterations in the figure as to be multiplied by 10.

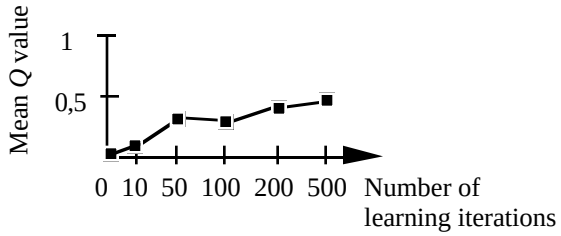


Figure 44. Mean *Q* values of all neurones during the learning phase. The global performance is increasing: the resulting behaviour is correct

In conclusion, we note that, unlike the multilayer perceptron implementation, the interpretation of the weights is possible. Moreover, if a correct behaviour is learned (i.e., only positive or null reinforcement values are experienced), then all neurones will code positive *Q* values. This last fact results in the optimisation of the stored knowledge.

7. COMPARISONS

Figure 45 presents the learning time, number of learning iterations and memory size of the two artificial neural network Q-learning implementations studied previously. A comparison with the other implementations of Q-learning has been done. Results displayed in figure 46 show that the self-organising map Q-learning implementation requires less memory and learns faster than all the others (by a factor of 40). The Q-KOHON implementation has also the best behaviour after learning, i.e., less negative reinforcements received than all the other implementations (fig. 39).

	Learning time (sec)	Nb. of learning it.	Memory size
Competitive MLP	190	500	84
Q-KOHON	40	100	176

Figure 45. Comparison of two different neural network implementations of the Q-learning in terms of learning time, number of learning iterations and memory size. The learning time is the time in seconds needed to synthesise an obstacle avoidance behaviour. It reflects the number of real world experiments required. The number of learning iterations is the number of updating of the neural network. The memory size is the number of connections in the neural network (one real value per connection).

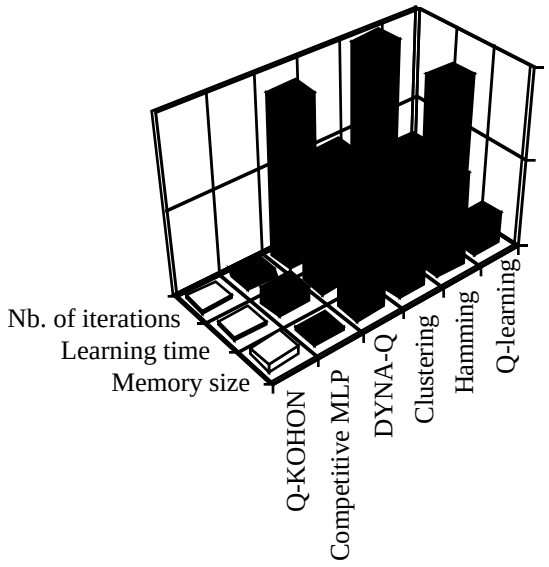


Figure 46. Comparison of several implementations of Q-learning on a task of learning an obstacle avoidance behaviour. The self-organising map Q-learning implementation (right) requires less memory and learns faster than all the others and more than 40 times faster than the basic Q-learning implementation (left).

8. DISCUSSION

Results obtained demonstrate that the neural generalisation allows us to speed up the Q-learning method. Starting from the QCON architecture, we have investigated the competitive multilayer perceptron implementation and the self-organising map implementation. The corresponding increase in performance seems to come from a localised coding on the output neurones and competition between the neurones. Therefore, other neural network models, like the ART (Adaptive Resonance Theory) network [18] which displays a partly-localised coding and general competition between the neurones, or the RBF (Radial Basis Function) network which enhances localised coding and general competition between the neurones, should be investigated. However, the unique property of neighbourhood of the Kohonen map, which influences the generalisation process through the continuity of the coding, may be very important for the quality of the results. In this case, we may have found with this implementation the 'ultimate' Q-learning implementation.

Reinforcement functions are usually hand-tuned and emerge after lots of experiments. The reinforcement learning algorithm task is to improve the cumulative reward over time. Despite a good learning phase (i.e., only positive *Q* values for all neurones of the self-organising map) the obtained behaviour does not always exhibit the expected behaviour

In our experiments, the learned behaviours show a large distribution of covered distances. Figure 47 displays four curves corresponding to four different experiments. A behaviour may exhibit a predilection for moving forward, or for moving backward, or for small movements, or for change in its policy during the experiment. There is also another behaviour, perhaps the most rewarding considering our reinforcement function describes in 1.2. It is: move forward to an obstacle, stop before the sum of the sensor values is above the threshold (2,90) and then move backward, and do it all again. This sequence of actions maximises rewards, but it is something different from the expected obstacle avoidance behaviour.

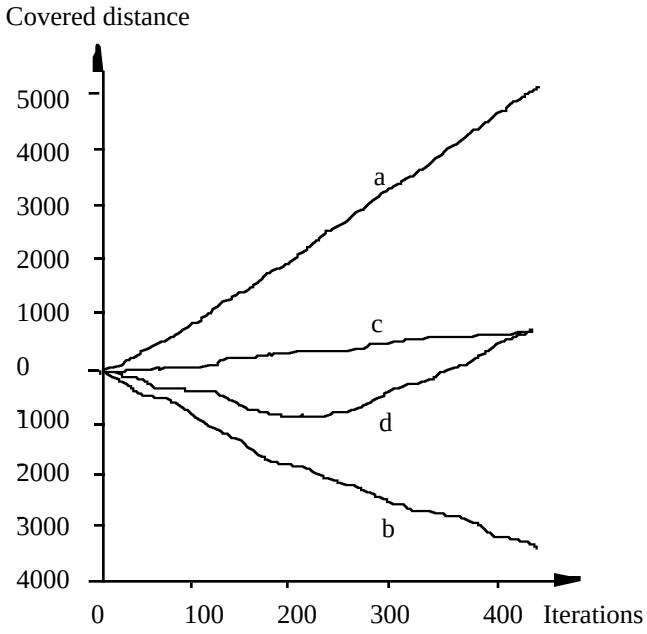


Figure 47. Distances covered by Khepera during four different experiments of learning an obstacle avoidance behaviour. Behaviour (a) displays a predilection for moving forward , (b) prefers moving backward, (c) prefers small forward movements, (d) changes its policy at the end of the learning phase (here 200 iterations).

8.1 Temporal credit-assignment problem

This result about the covered distances of several different experiments shows that having an effective Q-learning implementation is not sufficient to solve an application. In our particular case, the synthesised behaviour is a reactive behaviour. It does not integrate sequences of actions. A solution would be to change the reinforcement function to allow the taking into account of sequences of actions. Equation 1 (fig. 8) will then be used to solve the temporal credit assignment problem. However, its efficiency is strictly limited to short sequences of actions. The Q-learning implementation needs a concept of

historical states to deal with large sequences of actions. Temporal self-organising map and recurrent neural networks [21] appear to be good candidates to this end.

The description of the reinforcement function is of tremendous importance and research must be done for the design of reinforcement functions. We have started some work in this direction and proposed to add to the learning system an external module containing generic forbidden sequences of actions [22]. Experiments with the mobile robot Khepera demonstrate that it is possible to constrain the synthesised behaviours to forward avoidance behaviours.

The environment of our experiments is stationary; that is, the probabilities of making state transition, or receiving specific reinforcement signals, do not change over time. One may decide that this assumption makes our experiments trivial in that, after all, operation in non-stationary environments is one of the motivations for building learning systems [23]. In fact, the methods described here are effective in slowly varying non-stationary environments, in particular if learning does not stop and if there is always a probabilistic action selection function.

As expressed in section 1.1, obstacle avoidance in nothing else than a simple, pedagogical task for mobile robotics, aimed at allowing comparisons between reinforcement learning implementations. It is very difficult to achieve a solution with tabula rasa learning techniques (i.e., pure reinforcement learning). Bias must be incorporated to allow effective learning solutions, like: shaping, local reinforcement signals, imitation, problem decomposition, reflexes, etc.

Shaping: start with very simple tasks, then increase the difficulty. This method is particularly useful with supervised learning approaches [24].

Local reinforcement signals: whenever possible, give local reinforcement signals, rewarding local actions (not far from immediate reinforcement signals) [25].

Imitation: learn by watching an other agent. The problem is to get the first agent to perform the task. Today, this first agent must be human controlled in a task of obstacle avoidance [26].

Problem decomposition: decompose complex behaviour into a collection of simpler ones, and provide useful reinforcement signals for each one. This decomposition technique is used in almost all successful robotics reinforcement learning experiments [8].

Reflexes: [27] provide the robot with a set of reflexes, i.e., initial knowledge that will allow a better exploration of the search space (figure 48). This initial knowledge can be ameliorated through learning.

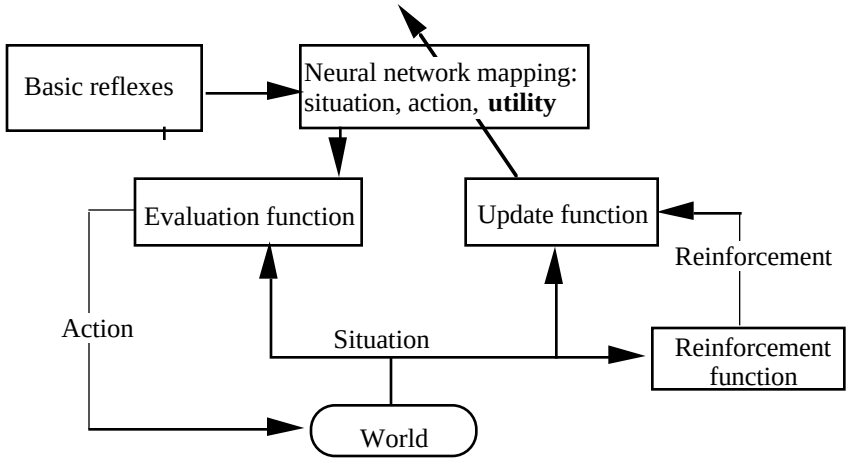


Figure 48. Learning from Basic Reflexes. This method utilises a set of basic reflexes every time its connectionist controller fails to generalise correctly its previous experience to the current situation, i.e., the evaluation function does not find an input neurone matching the current situation. The connectionist controller associates the selected reflex with the situation in one step. The sensory situation is represented by a new unit of the network and the selected reflex, or situation-action pair, is coded into the network weights. This new association is tuned subsequently through reinforcement learning. The neural network gets control more often as the robot explores the environment.

9. CONCLUSION

In this paper, we have presented the results of research aimed at improving reinforcement learning through the use of neural network implementations. The same real robot, environment and reinforcement function are used for all the experiments described in the paper. After a brief description of the Q-learning method, we have pointed out the need for generalisation. Several enhancements, including the Hamming distance, statistical clustering and Dyna-Q were briefly described. The results obtained with these implementations are interesting, but still insufficient at least in our experiment of learning an obstacle avoidance behaviour for the miniature mobile robot Khepera. Therefore, we have studied neural network implementations of Q-learning for their generalisation properties and limited computer memory requirements. An ideal neural implementation is proposed. It helps to understand the current limitations of the today implementations, like QCON (a multilayer perceptron implementation of Q-learning). A competitive multilayer

perceptron implementation is then proposed that allows better generalisation. This increase in performance seems to be the result of a localised coding on the output neurones and competition between neurones. A second neural implementation that takes full advantage of these two attributes is then proposed using a self-organising map. Results obtained on a task of learning an obstacle avoidance behaviour for the miniature mobile robot Khepera show that this last implementation is very effective.

It is our opinion that with a self-organising map implementation, Q-learning is no more a research area but a tool at the disposition of the engineer to develop applications in autonomous robotics. However, as discussed in section 8, there is still research to conduct in the Q-learning domain. In particular, the design of reinforcement functions is probably the most difficult part in the development of a reinforcement learning application. To deal with more complicated behaviour synthesis than the reactive behaviour chosen here, it is necessary to develop new Q-learning implementations. A concept of historical states is needed to deal with large sequences of actions. Temporal self-organising map and recurrent neural networks appear to be promising candidates to this end.

9. 1 Acknowledgements

We thank all the K-Team members (LAMI-EPFL, Switzerland) for their interest in this research and the use of one of the first Khepera robots.

10. REFERENCES

[1] F. Mondada, E. Franzi & P. Ienne, "Mobile Robot Miniaturisation: A Tool for Investigation in Control Algorithms," *Third International Symposium on Experimental Robotics*, Kyoto, Japan, October 1993.

[2] W. G. Walter, "An Imitation of Life," *Scientific American*, 182(5), 42-45, May 1950.

[3] V. Braitenberg, *Vehicles: Experiments in synthetic psychology*, MIT Press, 1986.

[4] L. Kaelbling, *Learning in embedded systems*, MIT Press, 1993.

[5] O. Holland & M. Snaith, "Extending the adaptive heuristic critic and Q-learning: from facts to implications," *Artificial Neural Networks, 2*, I. Alexander and J. Taylor (Eds.) Elsevier Science Publishers, 599-602, 1992.

[6] C. J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. thesis, King's College, Cambridge, England, 1989.

[7] S. Sehad & C. Touzet, "Reinforcement Learning and Neural Reinforcement Learning," *ESANN 94*, Editor M. Verleysen, D-Facto publication, Brussels, April 1994.

[8] M. Colombetti, M. Dorigo and G. Borghi, "Behavior Analysis and Training - A Methodology for Behavior Engineering," Special Issue on Learning Autonomous Robots, M. Dorigo Guest Editor, *IEEE Transactions on Systems, Man and Cybernetics*, SMC-part B, Vol. 26, No. 3, 365-380, June 1996.

[9] S. Mahadevan & J. Connell, "Automatic Programming of Behavior-based Robots using Reinforcement Learning," *Artificial Intelligence*, 55, 2, pp. 311-365, July 1991.

[10] R.S. Sutton, "Reinforcement Learning Architectures for Animats," *Proceedings of the First International Conference on Simulation of Adaptive Behavior, From Animals to Animats*, Edited by J-A Meyer and S.W. Wilson, MIT Press, pp. 288-296, 1991.

[11] A. G. Barto, R. S. Sutton & C. W. Anderson, " Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13: 834-846, 1983.

[12] A. G. Barto & P. Anandan, "Pattern Recognizing Stochastic Learning Automata," *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15 : 360-375, 1985.

[13] D. Rumelhart, G. Hinton & R. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing*, Vol. 1, D. Rumelhart & J. Mc Clelland Eds. Cambridge, MIT Press, pp. 318-362, 1986.

[14] L-J. Lin, "Reinforcement Learning for Robots Using Neural Networks," PhD thesis, Carnegie Mellon University, Pittsburgh, CMU-CS-93-103, January 1993.

[15] D. Ackley & M. Littman, "Interactions Between Learning and Evolution," *Artificial Life II*, SFI Studies Sc. Complexity, vol.X, C. G. Langton & Co Eds. Addison-Wesley, 487-509, 1991.

[16] C. Touzet & N. Giambiasi, "Application of Connectionist Models to Fuzzy Inference Systems", in Parallelization in Inference Systems, *Lectures Notes in Artificial Intelligence 590,* B. Fronhöfer & G. Wrightson Eds., Springer Verlag, April 1992.

[17] T. Kohonen, *Self-Organisation and Associative Memory*, Springer-Verlag, Berlin, 1984.

[18] G. A. Carpenter & S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network," *Proc. IEEE*, March 1988.

[19] J. S. Albus, "A new approach to manipulator control: the Cerebellar Model Articulation Controller (CMAC)," *Trans. ASME*, September 1975.

[20] R. A. Mc Callum, "Using transitional proximity for faster reinforcement learning," *Proc. of the Ninth International Conference on Machine Learning*, Morgan Kaufman (GB), 1992.

[21] C. Touzet & N. Giambiasi, "The Connectionist Sequential Machine: a General Model of Sequential Networks", *Australian Conf. on Neural Networks*, in Canberra, P. Leong & M. Jabri Eds. Sydney University Electrical Engineering, NSW 2006, Australia, February 1992.

[22] C. Touzet, S. Sehad & N. Giambiasi, "Improving Reinforcement Learning of Obstacle Avoidance Behavior with Forbidden Sequences of Actions," *International Conference on Robotics and Manufacturing*, Cancun, Mexico, 14-16 June 1995.

[23] L. Kaelbling, M. Littman and A. Moore, Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research* 4 , 237-285, 1996.

[24] Dorigo M. & Colombetti M, "Robot Shaping: developping autonomous agents through learning," *Artificial Intelligence*, vol. 71, n° 2, 321-370, 1994.

[25] M. Mataric, "Reward function for accelerated learning," In Cohen W.W. & Hirsh H. (Eds), *Proc. of the 11th Intern. Conf. on Machine Learning*, Morgan Kaufman, 1994.

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

[27] J. del R. Millàn, "Rapid, Safe and Incremental Learning of Navigation Strategies," Special Issue on Learning Autonomous Robots, M. Dorigo Guest Editor, *IEEE Transactions on Systems, Man and Cybernetics*, SMC-part B, Vol. 26, No. 3, 408-420, June 1996.