Proceedings of the IV Brazilian Conference on Neural Networks - IV Congresso Brasileiro de Redes Neurais
pp. 025-030, July 20-22, 1999 - ITA, São José dos Campos - SP - Brazil

Experiments in Robot Control for an Instance-Based Reinforcement Learning Algorithm based on Prior Information

Carlos H. C. Ribeiro[1] and Elder M. Hemerly[2]
[1] Divisão de Ciência da Computação - sala 106
[2] Divisão de Engenharia Eletrônica - sala 173
Instituto Tecnológico de Aeronáutica
Praça Mal. Eduardo Gomes, 50
12228-900 São José dos Campos - SP, Brazil
E-mails: c.ribeiro@ieee.org, hemerly@ele.ita.cta.br

Abstract

Reinforcement learning techniques are based on experience acquisition. In most realistic applications, experience is time-consuming: it implies sensor reading, actuator control and algorithmic update, constrained by the learning system dynamics. In fact, the information crudeness upon which classical learning algorithms operate can make such problems too difficult and unrealistic: embedding of prior structural knowledge, learning of control laws (instead of low level control actions) and full use of experience are required if control of real systems is at stake. In this paper, we show how a robust formulation (with respect to convergence properties) of the Q-learning method that considers prior information about the structure of the state space can be combined with an instance-based technique as a mechanism to accelerate learning. We then demonstrate it both in simulation (with a statistical consistency test) and in a real robot for a guidance task defined by a combination of a predefined control law and learned action policies.

1. Introduction

Reinforcement learning (RL) methods have been used for the problem of learning to control stochastic dynamic processes from direct experimentation. Such problems are usually considered in the background of Markov Decision Processes: the learning agent successively applies actions a_t for each sequential state observation $\mathbf{x}_t$, and receives an associated instant reward $r_t = r(\mathbf{x}_t, a_t)$. The goal of this experimentation process is finding out (through a learning method) an optimal policy of actions that maximises an expected cumulative reward. The problem is related to Dynamic Programming (DP) [1], with the basic difference being that, for RL, there is no prior model from which transition probabilities can be directly obtained. Nonetheless, former experience can be used to create an internal model from which additional learning updates can be carried out.

The best studied RL method is the Q-learning algorithm [2], in which the learning agent successively updates estimates $Q_t(\mathbf{x}_t, a_t)$ of action values $Q(\mathbf{x}_t, a_t)$, stored in a look-up table. Each action value represents the expected cost incurred by the agent when taking action a_t at state $\mathbf{x}_t$ and following an optimal policy thereafter. Formally, at time t the agent:

1. Visits state $\mathbf{x}_t$ and selects an action a_t.
2. Receives the reinforcement r_t and observes the next state $\mathbf{x}_{t+1}$.
3. Updates $Q_t(\mathbf{x}_t, a_t)$ according to:

$$\Delta Q_t = \alpha_t[r_t + \gamma \hat{V}_t(\mathbf{x}_{t+1}) - Q_t(\mathbf{x}_t, a_t)] \quad (1)$$

where $\Delta Q_t = Q_{t+1}(\mathbf{x}_t, a_t) - Q_t(\mathbf{x}_t, a_t)$, α_t is a learning rate and $\hat{V}_t(\mathbf{x}_{t+1}) = \max_a[Q_t(\mathbf{x}_{t+1}, a)]$ is the current estimate of the optimal expected cost $V^*(\mathbf{x}_{t+1})$. This method generates estimates Q_t that converge to Q, provided α_t satisfies $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$, and all the pairs $(\mathbf{x}, a)$ are visited infinitely often.

2. RL in Realistic Tasks

RL algorithms (Q-learning included) suffer from excessive conservatism as a single update for each state-action pair is performed for each experience undergone. For practical purposes this is a serious handicap, because real-life situations involve a very large number of states. Moreover, real-time delays and time constraints can further slow down the learning process.

In order to make better use of experience, a rather successful approach is to store former occurrences in a chain of instances, and then perform additional updates on them. However, performance improvement can still be not enough for learning tasks such as robot navigation, where there are system dynamics constraints which make learning extremely slow. Besides, wrong estimation of costs for a single state over a trajectory can completely jeopardize the obtained control policy for such problems. Some principles might be followed when applying RL methods to such hard tasks.

The first principle is that RL should not be applied at a too low level, because of the 'curse of dimensionality' [3]. By learning to coordinate control laws instead of control actions, the state space is greatly reduced. Furthermore, basic behaviours (minimal set of control laws that encapsulate sets of constraints so as to achieve particular goals [4]) are often easy to define and design. For example, consider the task of target approximation in an environment with obstacles. By defining a behaviour 'obstacle avoidance', whose activation triggers a simple obstacle avoidance controller, one can skip the problem of learning a control policy over the (many) states in the obstacle region, concentrating on more difficult tasks. For learning purposes, all what must be done with respect to obstacle avoidance is to provide a negative reinforcement every time the corresponding controller is activated. This procedure was employed in the experiments described in section 4.

Secondly, prior knowledge must be embedded. However, inclusion of additional information should not alter the robust properties of the algorithm, specially with respect to convergence guarantees. We consider two kinds of information embedding. The first is the adaptation of learning parameters, depending on the situation. For Q-learning, this would correspond to setting different learning rates or temporal discount factors, as appropriate. Clearly, this can violate convergence requirements. For instance, the learning rate for Q-learning must follow a $1/t$ intensity time variation, if optimality is required [5]. However, for practical purposes the possible advantages gained by proper selection of parameters may compensate for a loss of mathematical robustness, specially because such changes correspond to small structural modifications, whose qualitative implications are usually easy to assess. A second possible modification in order to embed prior information corresponds to more effective changes in the standard algorithm. This may correspond to the inclusion of additional parameters to encode the additional information. In this case, requirements for convergence are more important, because the effect of these extra parameters on the standard method can be overwhelming.

In what follows, we present a modified formulation of Q-learning that involves more than a single update per iteration, allowing for generalisation based on the spreading of the information derived from individual experiences. We show how this formulation is suited for embedding knowledge about the action value surface (action values or their respective rates of variation). Finally, we combine this technique with a model-based RL algorithm and describe experiments that show how a combination of prior information use (additional parameters from the modified algorithm and appropriate temporal discount factors) and iterative model construction can lead to good results in a real robotic task.

3. Generalising Q-Learning

Let us consider a more general version of the Q-learning algorithm, as follows. At time t, the agent:

1. Visits state $\mathbf{x}_t$ and selects an action a_t.
2. Receives the reinforcement r_t and observes the next state $\mathbf{x}_{t+1}$.
3. Update the action values for *every* state-action pair $(\mathbf{x}, a)$ according to:

$$\Delta Q_t^g = \alpha_t \sigma_t(\mathbf{x}, a)[r_t + \gamma \hat{V}_t(\mathbf{x}_{t+1}) - Q_t^g(\mathbf{x}, a) + K(\mathbf{x}, a)] \quad (2)$$

where $\Delta Q_t^g = Q_{t+1}^g(\mathbf{x}, a) - Q_t^g(\mathbf{x}, a)$, $K(\mathbf{x}, a)$ is an arbitrary bounded function, and $0 \leq \sigma_t(\mathbf{x}, a) \leq 1$ is the *spreading function*, which forces updates for pairs $(\mathbf{x}, a)$ possibly not involved in the experience at time t. The standard Q-learning update (1) corresponds to equation 2 with $\sigma_t(\mathbf{x}, a) = \delta(\mathbf{x}, \mathbf{x}_t)\delta(a, a_t)$, where $\delta(x, y)$ is defined as $\delta(x, y) = 1$ if $x = y$, otherwise $\delta(x, y) = 0$, and $K(\mathbf{x}, a) = 0$ (the identically null function).

The algorithm above is a generalisation of the standard Q-learning method in which a single experience can update more than a single action value. It converges to optimality, provided a) the standard conditions for convergence of Q-learning are met, b) the function $K(\mathbf{x}, a)$ is such that $K(\mathbf{x}_t, a_t) = 0$, and c) the spreading function converges to $\delta(\mathbf{x}, \mathbf{x}_t)\delta(a, a_t)$ at least as quickly as α_t converges to zero. A proof of convergence can be found in [6]. Roughly, this proof consists in identifying the algorithm above as an instance of a class of techniques that try to approximate a DP operator using a time-varying operator. This is justified by a theorem (whose proof is in [7]) that eliminates the burden of having to prove convergence of a cost functions V_t to V^*, by saying that it suffices to show that — given some conditions — the time-varying operator approximates a DP operator at a fixed point. By analysing our algorithm under the light of this theorem, it is not difficult to see that it does satisfies its conditions, and that it can be seen as a perturbed version of Q-learning, with a perturbation term that is of second-order with respect to the update term.

The general formulation can be adapted to versions of Q-learning that considers prior information about relative values or rates of variation of action values. For the sake of simplifying notation and without loss of generality, let us consider the one-dimensional problem ($\mathbf{x} = x$) with spreading limited to the state space. If we make $K(x, a) = 0$ (the identically null function), then we have the following algorithm:

$$\Delta Q_{t+1}^s = \alpha_t \sigma_t(x, a_t)[r_t + \gamma \hat{V}_t(x_{t+1}) - Q_t^s(x, a)] \quad (3)$$

We call this variant the QS algorithm (from Q-learning combined with Spreading).

Another variation corresponds to making $K(x, a) = (x - x_t)\hat{D}_{x_t}$, where $\hat{D}_{x_t}$ is a partial derivative estimate

$\frac{\partial \widehat{Q^*(x,a)}}{\partial x}$ for $x = x_t$. Notice that $\delta_t K(x,a) = \delta_t(x - x_t)\hat{D}_{x_t}$ is zero for all x, and therefore convergence (under the conditions stated in the last section) is guaranteed. The corresponding algorithm would be

$$\Delta Q^s_{t+1} = \alpha_t \sigma_t(x, a_t)[r_t + \gamma \hat{V}_t(y_t) - Q^s_t(x, a_t) + (x - x_t)\hat{D}_{x_t}] \quad (4)$$

4. Experimental Results

We used a Khepera robot [8] as the platform for our experiments. The learning goal is to control the robot for a combined obstacle avoidance and target approximation task.

4.1. Methodology

Figure 1 shows the setup for the experiments.

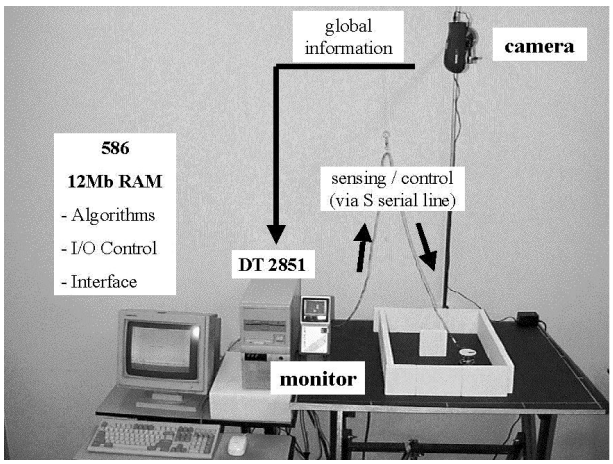


Figure 1: Setup for the experiments on robot learning.

The Khepera robot is controlled by low level commands sent via the computer's serial port (Pentium 586, 12Mb RAM). These commands are translations of high level instructions for wheel speed selection or position control, whose syntax is defined in the Khepera simulator [8]. Sensor readings for obstacle detection are carried out similarly. The robot can execute three actions: move forward a fixed distance (corresponding to the setting of a position controller based on dead-reckoning), turn left ninety degrees or turn right ninety degrees. The turning actions are set by direct speed control applied to the wheels, with an orientation error feedback computed with the help of the camera. This camera is interfaced to the computer via a Data Translation DT2851 B/W video capture board. The robot has on its top a white disk with a black direction mark that allows for orientation estimation with respect to a fixed Cartesian system defined by orthogonal axes fixed to the camera. Apart from providing feedback for control of the robot turns, the camera also indicates its position with respect to a rectangular grid. Notice that the use of global positioning information does not guarantees that the process is Markovian, because there are errors caused both by mechanical imperfections on the robot (generating uncertainties on measurements made by the wheel encoders for position control) and by sampling requirements between consecutive camera readings (producing errors of up to ten degrees on orientation estimation). Effects produced by the introduction of these imperfections are further discussed at the end of section 4.2.

The operation cycle for the learning algorithms is as follows:

1. Acquire and process image, read sensors.
2. Update learning parameters.
3. Perform control.

Because of the high cost of experience in real robot navigation learning tasks, we based our studies on the Dyna-Q algorithm [9]. Dyna-Q basically implements Q-learning, but apart from the action value tables (whose positions correspond to the rectangular grid positions of the camera image), it also stores a collection of past instances (tuples $\langle \mathbf{x}_t, r_t, a_t \rangle$) visited by the learning robot. For each learning step, Dyna-Q not only performs the standard Q-learning update for the current experience, but also for a collection N of past experiences, randomly chosen. We incorporated the spreading mechanism into Dyna-Q, creating a variant we call Dyna-QS, which was then tested in the navigation task. In order to assess the possible benefits of the inclusion of prior information, we also tested standard Dyna-Q under the same conditions.

The environment is a $54cm \times 37cm$ rectangular area, with a $10cm \times 2cm$ obstacle positioned as illustrated in figure 2. The camera captures the image of the environment in a 512×480 pixels frame, roughly aligned with the horizontal and vertical axes of the rectangular environment.

The robot must learn an action policy that avoids the use of a previously designed controller for obstacle avoidance (which when activated produces a -1 reinforcement), whilst at the same time trying to approach the target (a simulated light source which produces positive reward proportional to its proximity from the learning agent).

The learning parameters were set as follows:

- Discretisation levels (x,y): $(10, 16)$.
- Reinforcements: For activation of obstacle avoidance controller $r = -1.0$, with priority over reinforcement rewards obtained by target approximation. For target approximation, $r = 1.0$ (maximum) at target, then a decreasing linear function of the distance from target up to the distance of 450 pixels. Otherwise, $r = 0.0$
- Light source position (x,y) in pixels from bottom left corner: $(470, 280)$.

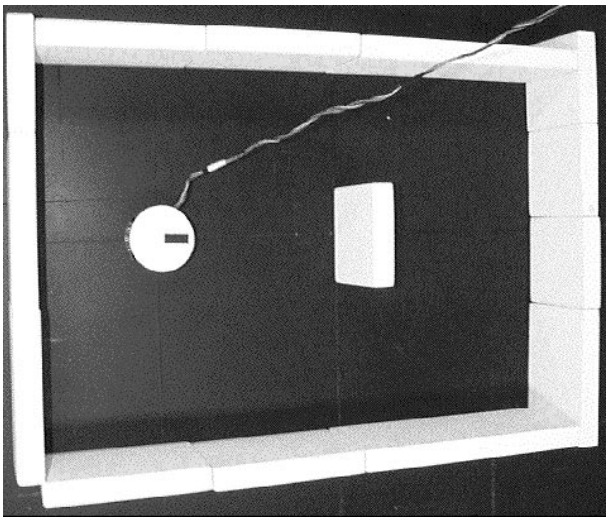


Figure 2: The learning environment. The white disk with mark (black rectangle) on the top of the robot facilitates calculation of positioning and orientation. The cable for communication between the robot and the host computer is also visible. The robot is located on the reference position at the start of a training course.

- Learning parameters: $\alpha = 0.9$, $\gamma = 0.1$ if obstacle avoidance controller is active, $\gamma = 0.9$ otherwise.We used a low value for obstacle avoidance because it should not imply keeping a long distance from obstacles, but only avoiding hitting them.
- Number N of past experiences revisited per iteration: 40.

The robot was always started from a reference position (Figure 2), with a learning protocol divided in two phases: in the first phase, it trains under its so far learned action policy, but with a 20% chance of choosing an explorative random action. During this phase, Dyna-QS produces spreading on the neighbourhood (closest 4 states) of the instantly visited state. Past experiences undergo standard Dyna-Q updates. In the second phase, the robot is again released from the same reference position and under the same action policy, but this time no spreading takes place. The learning phases are terminated once the robot explores the region close to the target. This is defined by a 50 steps additional training, once it reaches a position from which $r_t > 0.6$. Computing the overall learning time for the first two phases is a practical measure of the learning speed of the algorithm. Assessing the temporal variation of the expected reward and analysing the resulting trajectory during operation are measures of quality for the learned action policy.

4.2. Results

In order to give statistical consistency and motivation to our experiments, we initially compared Dyna-Q and Dyna-QS in a simulated experiment. The task was similar to the one used in the real setup. Twenty training courses for each algorithm were carried out, and a test of the hypothesis that the average number of steps ν_{QS} for Dyna-QS was smaller than the average number of steps ν_Q for Dyna-Q was performed assuming normal distributions and using sample variances. The results are summarized in Table 1. Notice that Dyna-QS learned trajectories much faster than Dyna-Q, due to its generalisation capabilities. The improvement is particularly significant in the second phase of training (no spreading taking place), showing that the generalisation performed in the first phase produced an action value space from which fine tuning (performed by Q-learning) could be carried out efficiently.

Table 1: Average number of steps and confidence level of hypothesis for simulated experiment.

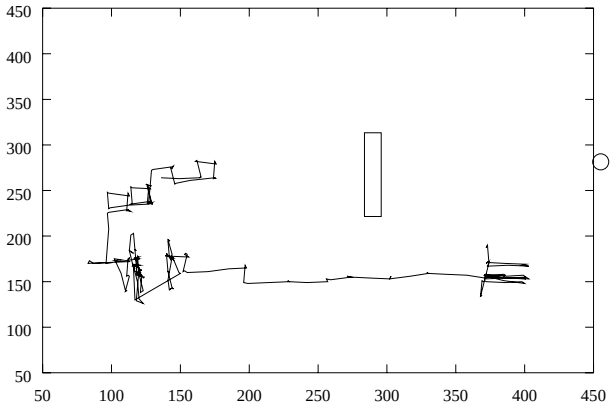
	phase 1	phase 2	total
ν_{QS}	757.0	251.2	1008.2
ν_Q	1201.0	687.5	1888.5
conf. level	> 0.95	> 0.94	> 0.95

Motivated by the results above, we ran five training courses in the real system, using the setup and parameters defined in the last section. Likewise, it turned out that Dyna-QS usually learned a trajectory much faster than Dyna-Q. It took an average 463.0 steps (226.2 steps in the first phase, 236.8 steps in the second phase) to reach $r_t > 0.6$, whilst Dyna-Q took an average 626.0 steps (respectively 372.6 and 253.4 steps in the first and second phases). Considering that each iteration step takes approximately $4.5sec$ real time to complete, this represents a $34min42sec$ average learning time for Dyna-QS against a $47min$ average learning time for Dyna-Q. Notice that although improvement was observed both in the simulated and in the real robot, there are quantitative differences in the results (number of steps) because of a) intrinsic modeling differences between simulation and practical realisation and b) different mechanisms used for controlling forward movement, which made the simulated robot move a relatively smaller distance than the real one for each iteration step.

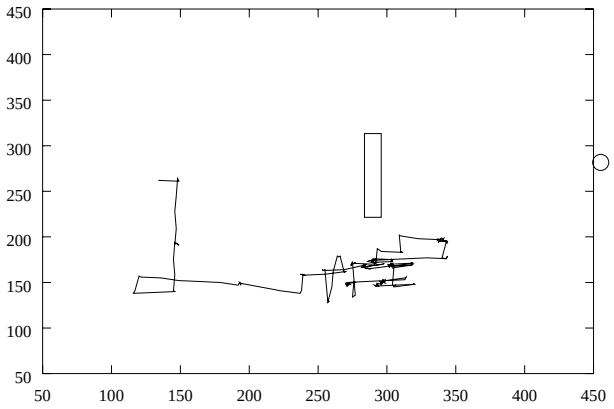
Figure 3 shows trajectories followed by the real Dyna-QS robot after the first, second and operational phases, respectively. Notice that the robot clearly managed to combine the obstacle avoidance and target approximation behaviours, though not in an optimal way. In particular, the robot does not get too close to the target position, partly because of the possibility of hitting the corresponding wall, and partly because it has been undertrained in that region (remember that training is interrupted 50 steps after the robot reaches $r_t > 0.6$ for the first time).

Figure 4 shows a typical average reward curve for the Dyna-QS learning agent, once it finishes training. It was obtained by averaging 5 realisations of trajectories obtained under the learned greedy policy.

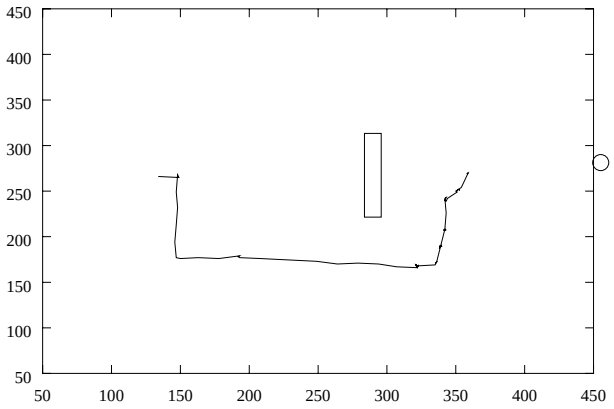
Typically, a learned trajectory is composed by piece-



(a)



(b)



(c)

Figure 3: Trajectories followed by the real Dyna-QS robot after the (a) first, (b) second and (c) operational phases. The small white circle on the right indicates the position of the light source, and the white rectangle indicates the position of the obstacle.

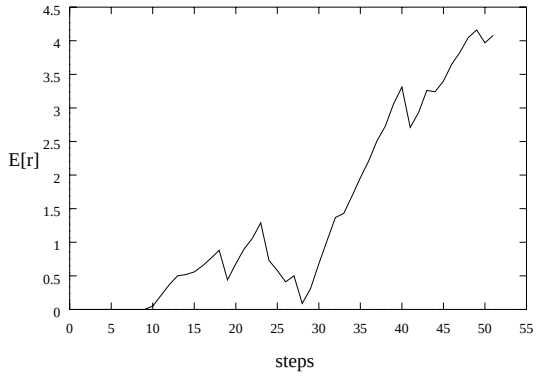


Figure 4: Average expected reward over 5 operation courses of the learned greedy action policy for the real Dyna-QS robot.

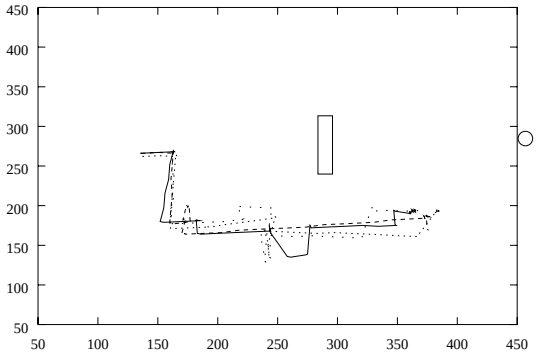


Figure 5: Trajectory variations among 4 realisations of a trained action policy for the real Dyna-QS robot.

wise suboptimal paths (defined by metastable policies, as pointed out in [2]), not necessarily linked to each other because of insufficient training. Use of prior information embedded in the spreading function helped Dyna-QS to connect these learned trajectories (and thus to make it learn the overall task faster), but a 10% action noise added to the learned policy proved to be helpful. The effect of the spreading function in accelerating learning is clear, as the robot managed to approach the target region already at the end of the first phase of training. In fact, spreading acts by improving learnability on the smoother parts of the action value space [10], whilst training under standard RL (in our case, Dyna-Q in the second phase) corrects generalisation mistakes and performs learning in 'rough' regions (near obstacle and walls).

Although the robot manages to reach a position close to the target once it is trained, there may be some variation among followed trajectories, as shown in figure 5. This is caused both by noise and bias introduced by wheel rotation encoding, failed orientation control (observe that the robot does not move along perfectly orthogonal paths), and loss of Markovian characteristics caused by discretisation of the state space.

5. Related Literature

Experience generalisation is related to state aggregation methods analysed by some authors [11, 12]. In particular, it can be shown [13] that Q-learning acting on a set of aggregate states converges, provided a persistently exciting action policy is used. However, the set of action values asymptotically reached will depend on the limit distribution $P^{\infty}(\mathrm{x})$ defined by this policy. The use of a decreasing spreading function eliminates this problem because it guarantees action value estimates that asymptotically approach the correct values, independently of the followed action policy.

The use of RL in autonomous robot control has been gaining space in the last few years. Some of the main issues involved in real tasks are reported in [14]. The concept of learning based on behaviours instead of low level actuator control was proposed by Matarić in its PhD thesis [15], and later on this approach was adopted by other authors [16, 17]. Dorigo and Colombetti combined RL and Classifier Systems to produce highly autonomous robots [18], based on a formalisation of the behavioural approach into a methodology similar to software engineering [19]. The idea of using behaviours not only for state space reduction but also as practical requirements for the safeness of the learning processes was proposed by Millan [20], who devised a connectionist architecture capable of combining fast learning from basic reflexes and high tolerance to sensor noise.

6. Summary

This paper demonstrated the applicability of a more general, convergent formulation for the tabular implementation of the Q-learning algorithm, that considers available information about the structure of the cost space. Experiments in a robotic navigation and guidance task showed that combining this method with an iterative model construction technique can accelerate learning in real time tasks, demonstrating the importance of extracting as much information as possible from every new experience undertaken by the agent in autonomous learning problems.

7. Acknowledgements

The authors are grateful to FAPESP (grant 97/07630-5) and CNPq (grant 300158/95-5) for supporting the work reported in this paper. Thanks are also due to Dr. Walter F. Lages for providing the Linux driver for the DT2851 board.

References

[1] R. Bellman. *Applied Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.

[2] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

[3] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, Belmont, Massachusetts, 1995.

[4] M. J. Matarić. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, May 1994.

[5] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.

[6] C. H. C. Ribeiro. Safe inclusion of information about rates of variation in a reinforcement learning algorithm. In A. P. Braga and T. B. Ludermir, editors, *Vth Brazilian Symposium on Neural Networks - SBRN'98*, volume 1, pages 2–7. IEEE Computer Society, 1998.

[7] M. L. Littman and C. Szepesvári. A generalized reinforcement learning model: Convergence and applications. In *Procs. of the Thirteenth International Conf. on Machine Learning (ICML'96)*, pages 310–318, 1996.

[8] K-Team S.A., Ch. de Vuasset, CP 111, 1028 Préverenges, Switzerland. *Khepera User Manual*, May 1998.

[9] R. S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Procs. of the 7th International Conf. on Machine Learning*, pages 216–224, 1990.

[10] C. H. C. Ribeiro. *Aspects of the Behaviour of a Learning Agent in Control Tasks*. PhD thesis, University of London, May 1998.

[11] J. N. Tsitsiklis and B. V. Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[12] S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 361–368. MIT Press, 1995.

[13] C. Szepesvári and M. L. Littman. Generalized markov decision processes: Dynamic-programming and reinforcement-learning algorithms. CS-96-11, Brown University, Department of Computer Science, Brown University, Providence, Rhode Island 02912, 1996.

[14] R. A. Brooks and M. J. Mataric. Real robots, real learning problems. In J. H. Connell and S. Mahadevan, editors, *Robot Learning*, chapter 8, pages 193–213. Kluwer Academic Publishers, 1993.

[15] M. J. Matarić. A distributed model for mobile robot environment learning and navigation. Master's thesis, Massachusetts Institute of Technology, 1990.

[16] Z. Kalmár, C. Szepesvári, and A. Lorincz. Module-based reinforcement learning: Experiments with a real robot. *Machine Learning*. to appear.

[17] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[18] M. Dorigo and M. Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71:321–370, 1994.

[19] M. Colombetti, M. Dorigo, and G. Borghi. Behavior analysis and training - a methodology for behaviour engineering. *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, 26(3):365–380, 1996.

[20] J. del R. Millán. Rapid, safe and incremental learning of navigation strategies. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 26(3):408–420, 1996 1996.