Autonomous Learning based on Cost Assumptions: Theoretical Studies and Experiments in Robot Control

CARLOS H. C. RIBEIRO & ELDER M. HEMERLY

TECHNOLOGICAL INSTITUTE OF AERONAUTICS

PRAÇA MAL. EDUARDO GOMES, 50

12228-900 SÃO JOSÉ DOS CAMPOS - SP, BRAZIL

Autonomous 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. The information crudeness upon which classical learning algorithms operate make such problems too difficult and unrealistic. Nonetheless, additional information for facilitating the learning process ideally should be embedded in such a way that the structural, well-studied characteristics of these fundamental algorithms are maintained. We investigate in this article a more general formulation of the Q-learning method that allows for a spreading of information derived from single updates towards a neighbourhood of the instantly visited state and converges to optimality. We show how this new formulation can be used as a mechanism to safely embed prior knowledge about the structure of the state space, and demonstrate it in a modified implementation of a reinforcement learning algorithm in a real robot navigation task.

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 often 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 closely related to Dynamic Programming [1], with the fundamental 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 probabilities can then be estimated or additional learning updates can be carried out.

The best studied RL method is the Q-learning algorithm [19], 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:

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

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

Many 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 complex learning tasks such as real-time robot navigation. In fact, wrong estimation of costs for a single state over a trajectory can actually completely jeopardize the obtained control policy for such problems. Certain principles must 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’ [2]. 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 [11]) are often easy to define and design. For instance, consider the task of target approximation in an environment with obstacles. By defining a behaviour ‘obstacle avoidance’, whose activation triggers a simple 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 actually employed in the experiments described below.

Secondly, prior knowledge must be embedded. However, inclusion of additional information ideally should not alter the robust properties of the algorithm, specially with respect to convergence guarantees. We consider two kinds of information embedding: first, 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 [19]. However, for practical purposes the possible advantages gained by proper selection of parameters may compensate for the 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, which encode the additional information. In this case, requirements for convergence are more important, because the effect of these additional parameters on the standard method can be overwhelming.

In what follows, we present and prove convergence of a modified formulation of the Q-learning algorithm which involves more than a single update per iteration, allowing for a generalisation mechanism based on the spreading of the information derived from individual experiences. Then 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 and iterative model construction can lead to good results in a real robotic navigation task.

2 Generalising Q-Learning

Let us consider a more general version of the Q-learning algorithm, as follows. Instead of the standard update Equation 1, consider updating the action values for *every* state-action pair $(\mathbf{x}, a)$, according to:

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

where $\Delta Q_t^g = Q_{t+1}(\mathbf{x}_t, a_t) - Q_t(\mathbf{x}_t, a_t)$ and $\sigma_t = \sigma_t(\mathbf{x}, a)$ is the *spreading function*, which forces updates for pairs $(\mathbf{x}, a)$ possibly not involved in the experience at time t . In practice, $0 \leq \sigma_t \leq 1$ is a measure of the similarity between states $\mathbf{x}_t$ and $\mathbf{x}$. The standard Q-learning update (1) corresponds to equation 2 with $\sigma_t = \delta(\mathbf{x}, \mathbf{x}_t) \delta(a, a_t)$, 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 therefore represents a generalisation of the standard Q-learning method in which a single experience (*i.e.*, a single loop of the algorithm) can update more than a single action value. The consequence of taking action a_t at state $\mathbf{x}_t$ is spread to other pairs $(\mathbf{x}, a)$ as if the real experience at time t actually was $\langle \mathbf{x}, a, \mathbf{x}_{t+1}, r_t \rangle$. As shown later, this general formulation can be related to an useful mechanism for embedding knowledge about relative values or rates of variation of action values.

3 Proof of Convergence

The proof of convergence for the algorithm above is based on a theorem that establishes a relationship between different kinds of stochastic approximations. The idea is to show that the algorithm relates to Q-learning in such a way that its convergence is guaranteed by the well-known convergence results of Q-learning itself.

3.1 Relating Stochastic Approximations

Finding an optimal action policy in Q-learning implies two approximations going on simultaneously: one of them approximates a DP operator, and the second one uses the approximations themselves to estimate the optimal costs (or action values). This contrasts with what would be a standard Robbins-Monro update for Q-learning:

$$\Delta Q_t = \alpha_t[r_t + \gamma V^*(\mathbf{y}) - Q_t(\mathbf{x}_t, a_t)] \quad (3)$$

which is naturally an unfeasible mechanism as V^* is not available in advance. Nevertheless, it might be convenient to relate our kind of stochastic approximation (characteristic of many RL algorithms) and simpler and better studied approximations such as equation 3. This would allow us to simplify or propose theorems for the more complicated cases by relating them to their simpler counterparts. First, however, it is important to review a theorem that relates these kinds of stochastic approximations.

Let $B(\mathcal{X})$ be the set of cost functions over $\mathcal{X}$ and $T : B(\mathcal{X}) \rightarrow B(\mathcal{X})$ be an arbitrary contraction mapping with fixed point V^* . For DP algorithms in general, a contraction mapping T (or T^q) is applied directly to successively approximate V^* (or Q). In model-free RL methods, such operator is not available and the agent must use its own experience to approximate it.

Consider then a sequence of operators $T_t : (B(\mathcal{X}) \times B(\mathcal{X})) \rightarrow B(\mathcal{X})$ and define $U_{t+1} = T_t U_t V$ where V and U_0 are arbitrary cost functions. We say that T_t approximates T at V with probability one uniformly over $\mathcal{X}$ if U_t converges to TV uniformly over $\mathcal{X}$.

We can interpret this in the following way: a ‘memory’ U_t is used as a help to make T_t approximate T . The equivalence between these operators is assessed by using a ‘test function’ V that subject either to T (through TV) or to T_t (through $T_t U_t V$) produces the same result.

Theorem 1 *Let T be an arbitrary mapping with fixed point V^* , and let T_t approximate T at V^* with probability one uniformly over $\mathcal{X}$. Let V_0 be an arbitrary cost function, and define $V_{t+1} = T_t V_t V_0$. If there are functions $0 \leq F_t((x)) \leq 1$ and $0 \leq G_t((x)) \leq 1$ satisfying the conditions below with probability one, then V_t converges to V^* with probability one uniformly over $\mathcal{X}$.*

1. For every $U_1, U_2 \in B(\mathcal{X})$ and $\mathbf{x} \in \mathcal{X}$,

$$|(T_t U_1 V^*)(\mathbf{x}) - (T_t U_2 V^*)(\mathbf{x})| \leq G_t(\mathbf{x}) |U_1(\mathbf{x}) - U_2(\mathbf{x})| \quad (4)$$

2. For every $U, V \in B(\mathcal{X})$ and $\mathbf{x}, \mathbf{y} \in \mathcal{X}$,

$$|(T_t UV^*)(\mathbf{x}) - (T_t UV)(\mathbf{x})| \leq F_t(\mathbf{x}) \sup_{\mathbf{y}} |V^*(\mathbf{y}) - V(\mathbf{y})| \quad (5)$$

3. For all $k > 0$, the product $\prod_{t=k}^n G_t(\mathbf{x})$ converges to zero uniformly in $\mathbf{x}$ as n increases;
4. There is $0 \leq \beta \leq 1$ such that for all $\mathbf{x}$ and large enough t , $F_t(\mathbf{x}) \leq \beta(1 - G_t(\mathbf{x}))$.

The important aspect of this theorem (whose proof is in [8]) is that it eliminates the burden of having to prove the convergence of the cost functions V_t to V^* . Instead, it says that it suffices to show that — given the conditions — the operator T_t approximates the operator T at the fixed point V^* .

3.2 Convergence Proof

First, operators T and T_t must be defined. Then, conditions 1 to 4 of theorem 3.1 must be verified. Finally, it must be checked if T_t approximates T with probability 1.

The operators T and T_t are respectively the Action Value operator

$$(T^q Q)(\mathbf{x}, a) = r(\mathbf{x}, a) + \gamma \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, a) \min_u Q(\mathbf{y}, u) \quad (6)$$

and the random operator sequence

$$(T_t^{qq} UV)(\mathbf{x}, a) = U(\mathbf{x}, a) + \alpha_t \sigma_t(\mathbf{x}, a) [r_t + \gamma \min_u V(\mathbf{y}, u) + K(\mathbf{x}, a) - U(\mathbf{x}, a)] \quad (7)$$

with $U, V : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{R}$. Equation 2 can then be written as $Q_{t+1}^g = T_t^{qg} Q_t^g Q_t^g$, and if the conditions of theorem 3.1 are satisfied then this process converges to the fixed point of T^q — namely, Q .

Let us then choose $G_t(\mathbf{x}, a) = 1 - \alpha_t \sigma_t(\mathbf{x}, a)$ and $F_t(\mathbf{x}, a) = \alpha_t \sigma_t(\mathbf{x}, a)$ which respectively satisfy conditions 1 and 2 of Theorem 3.1.

Condition 3 would imply $\prod_{t=k}^{\infty} (1 - \alpha_t \sigma_t(\mathbf{x}, a)) = 0$. Consider then this result [7]: if $0 < m_t < 1$ then $\prod_{t=k}^{\infty} (1 - m_t) = 0$ iff $\sum_{t=k}^{\infty} m_t = \infty$. In our case, this means that $\sum_{t=k}^{\infty} \alpha_t \sigma_t(\mathbf{x}, a) = \infty$ is required. However, if we assume $\sum_{t=k}^{\infty} \alpha_t \delta(\mathbf{z}, x_t) \delta(u, a_t) = \infty$ (mandatory to ensure convergence of the Q-learning algorithm [5]) and $\delta(\mathbf{z}, x_t) \delta(u, a_t) \leq C \sigma_t(\mathbf{x}, a)$ for some $0 < C \leq \infty$, then $C \sum_{t=k}^{\infty} \alpha_t \sigma_t(\mathbf{x}, a) \geq \sum_{t=k}^{\infty} \alpha_t \delta(\mathbf{z}, x_t) \delta(u, a_t) \geq \infty$, satisfying condition 3.

Also, notice that condition 4 is satisfied because $F_t(\mathbf{x}, a) = 1 - G_t(\mathbf{x}, a)$.

Now we must check if T_t^{qg} approximates T^q — the Q-learning operator of equation 6 — with probability 1. By definition, this is equivalent to showing that for a fixed $\hat{V}(\mathbf{y})$, T_t^{qg} approximates T^q at a fixed point. But as we already know that Q-learning converges to Q (and therefore its corresponding random operator T_t^q approximates T^q at Q), it suffices to show that for a fixed $\hat{V}(\mathbf{y})$

$$\Delta Q_t^g = \alpha_t \sigma_t [r_t + \gamma \hat{V}_t(\mathbf{y}) + K(\mathbf{x}, a) - Q_t^g(\mathbf{x}, a)] \quad (8)$$

and

$$\Delta Q_t = \alpha_t \delta_t(\mathbf{x}, a) [r_t + \gamma \hat{V}_t(\mathbf{y}) - Q_t(\mathbf{x}, a)] \quad (9)$$

converge to the same value. Let us fix an arbitrary pair $(\mathbf{x}, a)$ and denote by Q_t^g , Q_t , *etc.* the values of $Q_t^g(\mathbf{x}, a)$, $Q_t(\mathbf{x}, a)$, *etc.* If $\alpha_t = 0$ then neither Q_t^g nor Q_t changes, so we can consider $\alpha_t > 0$ for all t . Assume now that $\sigma_t - \delta_t = O(\alpha_t)$, *i.e.*, there is a bounded function B_t such that $\sigma_t - \delta_t = B_t \alpha_t$. Simple algebraic manipulation on equations 8 and 9 yields

$$\begin{aligned} Q_{t+1}^g &= Q_t^g + \alpha_t \delta_t [r_t + \gamma \hat{V}_t(\mathbf{y}) - Q_t^g] + \\ &\quad \alpha_t^2 B_t [r_t + \gamma \hat{V}_t(\mathbf{y}) + K(\mathbf{x}, a) - Q_t^g] + \\ &\quad \alpha_t \delta_t K(\mathbf{x}, a) \end{aligned} \quad (10)$$

If we select a function $K(\mathbf{x}, a)$ such that the product $\delta_t K(\mathbf{x}, a)$ is zero for all x , then

$$\begin{aligned} Q_{t+1}^g &= Q_t^g + \alpha_t \delta_t [r_t + \gamma \hat{V}_t(\mathbf{y}) - Q_t^g] + \\ &\quad \alpha_t^2 B_t [r_t + \gamma \hat{V}_t(\mathbf{y}) + K(\mathbf{x}, a) - Q_t^g] \end{aligned} \quad (11)$$

with a perturbation term $\alpha_t^2 B_t [r_t + \gamma \hat{V}_t(\mathbf{y}) + K(\mathbf{x}, a) - Q_t^g]$ that can be neglected for small learning rates. Thus, for this kind of $K(\mathbf{x}, a)$ equations 8 and 9 converge to the same value for a fixed $\hat{V}(\mathbf{y})$, and therefore the operator T_t^{qg} approximates T^q .

4 Embedding Prior Knowledge

The general formulation can be easily 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 ($\mathbf{x} = x$).

If we make $K(x, a) = 0$ (the identically null function), then we have the following update equation:

$$\Delta Q_t^s = \alpha_t \sigma_t [r_t + \gamma \hat{V}_t(y) - Q_t^s(x, a)] \quad (12)$$

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

Figure 1. Setup for the experiments on robot learning.

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 $\frac{\partial Q^*(x, a)}{\partial x}$ 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 update equation is now

$$\begin{aligned} \Delta Q_t^s &= \alpha_t \sigma_t [r_t + \gamma \hat{V}_t(y_t) - \\ &\quad Q_t^s(x, a) + (x - x_t) \hat{D}_{x_t}] \end{aligned} \quad (13)$$

5 Experimental Results

We used a Khepera robot [12] as the platform for our experiments. The learning goal is to control the robot for a combined obstacle avoidance and target approximation task.

5.1 Methodology

Figure 1 shows a schematic setup for the experiments.

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 instructions for wheel speed selection or position control, whose syntax is defined in the Khepera simulator [12]. Sensor readings for obstacle detection are carried out similarly. The robot can execute three actions: move forward a fixed distance (activation 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, which 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 (see Figure 2). Apart from providing the feedback for control of robot turns, the camera also indicates its position with respect to a rectangular grid. Differently from simulated experiments, the use of global positioning information does not guarantee that the process is first-order Markov, because there are imprecisions 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). In fact, these additional complications made it very difficult to transfer simulation results to the robot, and motivated us to carry out the learning process directly on the real system. Effects produced by the introduction of these imperfections are further discussed at the end of Section 5.2.

The operation cycle for the learning algorithms is: 1) acquire and process image and read obstacle and light sensors, 2) update learning parameters, and finally 3) perform control.

We based our studies on the Dyna-Q algorithm [15]. 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 the collection of past instances (tuples $\langle x_t, r_t, a_t \rangle$) 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 randomly chosen past experiences.

We incorporated the spreading mechanism into the Dyna-Q algorithm, 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 learning task consisted of a combined obstacle/wall avoidance and target (light source) approximation. The environment is a $54\text{cm} \times 37\text{cm}$ rectangular area, with a $10\text{cm} \times 2\text{cm}$ 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, whilst at the same time trying to approach

Figure 2. The learning environment. The cable for communication between the robot and the computer is visible.

the target.

We used 160 discretisation levels (10 x axis, 16 y axis); a $r = -1.0$ reinforcement for activation of obstacle avoidance controller, with priority over reinforcement rewards obtained by target approximation. For rewards, we have $r = 1.0$ (maximum) at target, then a decreasing linear function of the distance from target; a learning rate $\alpha = 0.9$; a discount factor $\gamma = 0.1$ if obstacle avoidance controller is active, $\gamma = 0.9$ otherwise; and finally $N = 40$ visitations to past experiences per iteration. The light source position (x,y) in pixels from bottom left corner is (470,280).

The robot was always started from a reference position (see 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 a random action, to allow for exploration of the state space. 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.

5.2 Results

For five training courses as defined above, 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 learning time for Dyna-QS against a 47min learning time for Dyna-Q.

Figure 3 shows trajectories followed by the 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).

Typically, a learned trajectory is composed by piecewise suboptimal paths (defined by metastable policies [19], 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 quickly approach the target region already at the end of the first phase of training. In fact, spreading acts by improving learnability of the smoother parts of the action value space [13], whilst training under standard RL (in our case, Dyna-Q in the second phase) corrects generalisation mistakes and performs learning in ‘rough’ regions of the action value space (regions around obstacle and walls).

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. 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

Figure 3. Trajectories followed by the 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.

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

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

space.

6 Related Literature

Experience generalisation is related to state aggregation methods in the context of function approximation for RL [18, 14]. In particular, it can be shown [17] that Q-learning 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 $P^\infty(x)$ defined by this policy. If the actions themselves depend on the estimated action values, convergence is not guaranteed [16]. The use of a decreasing spreading function eliminates this problem because it ensures action value estimates that asymptotically approach the correct values, independently of the 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 which make them particularly difficult are reported in [3]. The concept of learning based on behaviours instead of low level

actuator control was proposed by Mataric in its PhD thesis [10], and later on this approach was adopted by other authors [6, 9]. 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 [4].

7 Summary

This paper presented and demonstrated the applicability of a more general, convergent formulation of Q-learning algorithm, that considers available information about the structure of the cost space. Experiments in a robot navigation and guidance task showed that combining this method with a 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.

8 Acknowledgements

The authors are grateful to FAPESP (grant 97/07630-5) and CNPq (grant 300158/95-5). 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] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, Belmont, Massachusetts, 1995.
- [3] 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.
- [4] 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.
- [5] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.
- [6] Z. Kalmár, C. Szepesvári, and A. Lorincz. Module-based reinforcement learning: Experiments with a real robot. *Machine Learning*. to appear.
- [7] S. Karlin and H. M. Taylor. *A First Course in Stochastic Processes*. Academic Press, 1975.
- [8] 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.

- [9] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.
- [10] M. J. Matarić. A distributed model for mobile robot environment learning and navigation. Master's thesis, Massachusetts Institute of Technology, 1990.
- [11] M. J. Matarić. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, May 1994.
- [12] K-Team S.A., Ch. de Vuasset, CP 111, 1028 Prévèrèges, Switzerland. *Khepera User Manual*, May 1998.
- [13] C. H. C. Ribeiro. *Aspects of the Behaviour of a Learning Agent in Control Tasks*. PhD thesis, University of London, May 1998.
- [14] 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.
- [15] 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.
- [16] C. Szepesvári. *Static and Dynamic Aspects of Optimal Sequential Decision Making*. PhD thesis, József Attila University, Szeged, Hungary, May 1997.
- [17] 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.
- [18] J. N. Tsitsiklis and B. V. Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.
- [19] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.