Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Cognitive Systems Research 4 (2003) 319–337

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

Event-learning and robust policy heuristics

Action editor: Vasant Honavar

András Lörincz*, Imre Pólik, István Szita

Department of Information Systems, Eötvös Loránd University, Pázmány Péter sétány 1/C, H-1117 Budapest, Hungary

Received 1 April 2002; received in revised form 1 August 2002; accepted 1 September 2002

Abstract

In this paper we introduce a novel reinforcement learning algorithm called event-learning. The algorithm uses *events*, ordered pairs of two consecutive states. We define event-value function and we derive learning rules. Combining our method with a well-known robust control method, the SDS algorithm, we introduce Robust Policy Heuristics (RPH). It is shown that RPH, a fast-adapting non-Markovian policy, is particularly useful for coarse models of the environment and could be useful for some partially observed systems. RPH may be of help in alleviating the 'curse of dimensionality' problem. Event-learning and RPH can be used to separate time scales of learning of value functions and adaptation. We argue that the definition of modules is straightforward for event-learning and event-learning makes planning feasible in the RL framework. Computer simulations of a rotational inverted pendulum with coarse discretization are shown to demonstrate the principle.

Keywords: Reinforcement learning; Event-learning; Robust control; Continuous SDS controller

1. Introduction

In a common formulation of the reinforcement learning (RL) problem an agent improves its behavior by observing the outcomes of its own interactions with the environment. Several years ago Markovian decision problems (MDPs) were proposed as the model for the analysis of RL (Andreae, 1969; Witten, 1977; Watkins, 1989), and since then a mathematically well-founded theory has been constructed for a large class of RL algorithms. These algorithms are based on two basic dynamic-programming methods, namely the value- and policy-iteration algorithms (Watkins, 1989; Jaakkola, Jordan, & Singh, 1994; Tsitsiklis & Van Roy, 1996; Sutton, 1996; Szepesvári & Littman, 1999). The basic properties of most of the theoretical results are that they assume finite state- and action-spaces, discrete-time models in which the full description of the state was available. In many real-life problems, however, the state- and action-spaces are infinite (but see (Barto, 1978) for a discussion), usually non-discrete, time is continuous and the system's state is not fully known (the state is only partially observed). Although these interesting, yet theoretically more difficult cases were investigated by many researchers (see e.g. Littman, Cassandra, & Kaelbing, 1995; Singh, Jaakkola, & Jordan, 1995), no complete and theoretically sound solution has been found to date that is computationally tractable for large problems.

*Corresponding author. Tel.: +36-1-463-3515; fax: +36-1-463-3490.

E-mail address: lorincz@inf.elte.hu (A. Lörincz).

1389-0417/03/$ – see front matter
doi:10.1016/S1389-0417(03)00014-7

If the system's state is only partially observable, the application of such general learning algorithms becomes increasingly difficult (Littman, 1996).

Another problem of RL is the separation of learning the value function (e.g. state value function) from the adaptation of the dynamics. As an illustration, consider a robotic arm which has to lift a box to a given height. The initial and final states are given, but the weight of the box is unknown. The problem is that the estimation of the value function depends on the weight. A human easily adapts to the new situation, but most reinforcement learning methods re-learn the value function for every new box or, alternatively, must include the mass as a parameter into the learning process and need at least an approximate mass value as input. In turn, either re-learning or an increase of the dimensionality of the problem would be necessary for most RL methods. If we were able to execute actions by using an approximation of the correct value function (e.g., because the weight has changed), then the re-learn process could be postponed or avoided. Adaptation to the correct dynamics *during* task execution improves the chances of success.[1] Such an algorithm could be insensitive to the unmodeled perturbations of the dynamics of the system.

In this article we introduce a new RL algorithm called *event-learning* which deals with the said problems. In the ordinary settings, in each state an action (behavior) is selected, and our goal is to optimize this selection. In the new setting, in a given state a desired new state is selected, then a separate routine determines the control action that may or may not reach the desired state.

One of our motivations to explore event-learning comes from our desire to develop an RL framework, which:

1. Is capable of controlling real-world tasks that are difficult for standard RL techniques, tasks with incomplete state observation and continuous action spaces, where control must be stable despite of the perturbed system dynamics. This is different from most of the techniques in the RL literature, where the task has a discrete action space, and the objective is to learn a good solution under the assumption that the task dynamics are static.
2. The framework may promote planning abilities. The following analogy is given to explain the motivation: Optimization of state-dependent action selection can be interpreted as a *habit*, *conditioned reflex* or *shaped behavior*. In contrast, the selection of a desired next state in goal-oriented problems *without* explicit references to control actions seems more adequate for planning purposes.
3. The framework does not compromise attractive properties of RL, such as delayed reinforcement.

The paper is organized as follows. A short theoretical overview is given in Section 2. The descriptions of MDP and RL are followed by the definition of event-learning in Section 3. Section 4 describes the (non-Markovian) robust policy heuristics (RPH) that applies the Static and Dynamic State (SDS) feedback controller. In Section 5, properties of event-learning are discussed, with computational demonstrations partially outside the realm of the theory. Conclusions are drawn in Section 6. Some possible extensions of event-learning are also listed in this last section. Mathematical details are presented in Appendices A–C.

2. An overview of reinforcement learning

In this section we give a short overview of the Reinforcement Learning Problem. In one well-studied setting (see e.g. Sutton & Barto, 1998) the agent maximizes *the expected discounted reward*, i.e. $\Sigma_0^\infty \gamma^t r_t$, where r_t is the immediate reward at time step t, and $\gamma \in [0, 1]$ is the discount factor. If this maximization is made over a discrete-time, finite-state, finite-action environment, the theory of Markovian Decision Problems can be applied as the underlying mathematical model. A finite MDP is defined by the 4-tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $p(s, a, s')$ is the probability of getting to s' when taking

[1] We distinguish learning and adaptation. Learning concerns the improvement of the ratio of successful trials over unsuccessful ones given the success or failure of the individual trials. In contrast, adaptation is a *within trial* fast process.

action a in state s, and $r(s, a, s')$ is the expected immediate reward of this transition.

The task of the learning agent is to find a behavior (policy) that yields the most cumulated reward. In the usual formulation, the policy is described by a probability distribution function π: $\mathcal{S} \times \mathcal{A} \to R$ that determines the probability of each action in each state. A policy π is optimal under the *expected discounted total reward criterion* if, with respect to the space of all possible policies, π results in a maximum of expected discounted total reward for all states.[2]

The class of RL algorithms of our interest are variants of the value-iteration method: these algorithms gradually improve an estimate of the optimal value function via learning from interactions with the environment. The state-value function of a policy π in state x is defined as the expected discounted cumulated reward that can be gained by starting from state x and following policy π:

$$V^{\pi}(x) = \mathbf{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) | s_0 = x\right], \tag{1}$$

where $\mathbf{E}_{\pi}$ denotes expected value under policy π. The state-value function can be expressed recursively by

$$V^{\pi}(x) = \sum_{a \in A(x)} \pi(x, a) \sum_{y} p(x, a, y)(r(x, a, y) + \gamma V^{\pi}(y)) \tag{2}$$

for all states $x \in \mathcal{S}$, where $\Pr(a_t = a | s_t) = \pi(s_t, a)$, $\Pr(s_{t+1} = y | s_t, a_t) = p(s_t, a_t, y)$ $(t = 0, 1, 2, ...)$, and $A(x) \subset \mathcal{A}$ is the set of admissible actions in state x.

It is well known that there is a unique optimal value function, whereas there may be many optimal policies, e.g. by symmetry reasons. For a review on policy evaluation and policy improvement, see, e.g., (Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998).

Besides V^{π}, some other types of value functions can be defined as well. One example is the state–action-value function:

$$Q^{\pi}(x, a) = \mathbf{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) | s_0 = x, a_0 = a\right], \tag{3}$$

for all $x \in \mathcal{S}$ and $a \in \mathcal{A}$. $Q^{\pi}(x, a)$ has the meaning 'the expected cumulated discounted reward of taking action a in state x and then following policy π.' The state–action value function is especially useful, since it can be used for control without the need of building a model of the environment (i.e. there is no need to estimate the transition probabilities), which may prove to be advantageous in many applications. Clearly, the state–action-value function can be expressed in terms of V:

$$Q^{\pi}(x, a) = \sum_{y} p(x, a, y)(r(x, a, y) + \gamma V^{\pi}(y)), \tag{4}$$

while the reformulation of V^{π} in terms of Q^{π} is the following:

$$V^{\pi}(x) = \sum_{a \in A(x)} \pi(x, a) Q^{\pi}(x, a). \tag{5}$$

Combining these two equations, a recursive equation for Q^{π} appears, which is the base of applying a value-iteration algorithm:

$$Q^{\pi}(x, a) = \sum_{y} p(x, a, y)\left(r(x, a, y) + \gamma \sum_{v \in A(y)} \pi(y, v)\, Q^{\pi}(y, v)\right). \tag{6}$$

On-line history using policy $\pi(s, a)$ can be seen as a sampling of the probabilities, so Eq. (6) can be used to compute the value functions directly, without estimating the transition probabilities, e.g. by the following update rule (Rummery & Niranjan, 1996; Sutton, 1996):

$$Q^{\pi}_{t+1}(s_t, a_t) = (1 - \alpha_t(s_t, a_t))\, Q^{\pi}_t(s_t, a_t) + \alpha_t(s_t, a_t)(r_t + \gamma Q^{\pi}_t(s_{t+1}, a_{t+1})), \tag{7}$$

where r_t is the experienced reward at time t, actions are selected according to policy $\pi(s, a)$ and the learning rates $\alpha_t(s_t, a_t) \geqslant 0$ satisfy the usual Robbins–Monro type conditions. For example, one might set $\alpha_t(s, a) = \frac{1}{n_t(s, a)}$ where $n_t(s, a) = 1 + \mathrm{C}\{s_i = s,$

[2] Note that in most cases it suffices to maximize the cumulative reward without identifying an optimal policy for the whole state space, since there may be regions of the state space that the agent never visits.

$a_i = a | i = 0 \ldots t\}$,[3] but often in practice $\alpha_t(s, a) =$ const is employed which improves adaptation but no longer ensures convergence. The update is called SARSA update because it is using experienced **s**tate(t)–**a**ction(t)–**r**eward(t)–**s**tate(t+1)–**a**ction(t+1) tuples for evaluation. If policy π visits every states infinitely often, this algorithm is guaranteed to converge with probability one to the optimal value function (Singh, Jaakkola, & Littman, 2000). In the simplest case, ε-greedy policy is applied. In state x, this policy selects the action with highest value with probability $1-\varepsilon$ and a random action with probability ε. The ε-greedy policy attains a reasonable compromise between the *exploitation* of existing knowledge and the need for *exploration*. For a detailed description see, e.g., (Sutton & Barto, 1998).

3. Event-learning

Below we introduce *event-learning*, where in state x a new desired state y^d is selected (instead of selecting an action). This selection is also based on a value function, the *event-value function* (to be defined later in this section). Upon selecting a desired state, we need to solve the problem of 'getting there.' We will pass this problem to a lower-level controller, which operates independently of the upper-level process. This decision-decomposition can be formulated more formally: Policy π is decomposed into *event policy* π_E: $\mathcal{S} \times \mathcal{S} \to R$ and *controller policy* π_A: $\mathcal{S} \times \mathcal{S} \times \mathcal{A} \to R$, where $\pi_E(x, y^d)$ is the distribution of selecting y^d as new desired state in x, and similarly, $\pi_A(x, y^d, a)$ is the distribution of selecting control action a in state x in order to get to y^d (a may or may not be able to realize this transfer). Then $\pi(x, a)$ can be computed by marginalizing over y^d:

$$\pi(x, a) = \sum_{y^d} \pi_E(x, y^d)\, \pi_A(x, y^d, a). \tag{8}$$

In general, the policy π_A realized by the sub-level controller cannot always transfer the agent to y^d. However, our aim is to find a controller that performs well at least locally, i.e. for desired states that are in the neighborhood of x (if such a neighborhood is defined).

Note that from the point of view of the policy π_E, the output of the controller can be seen as a part of the environment, similarly to the transition probabilities $p(x, a, y)$.

The pair (x, y^d) is called the desired *event* (hence the name *event-learning* or *event-learning*). In general, any ordered pair of two states can be viewed as an event. Practically, an event occurs if it is made up by two consecutive states.

The algorithm of event-learning can be scheduled as follows. For a given initial state s_1 select a desired state s_1^d and then pass the formed desired event to the controller. The controller selects an appropriate action, then this action results in the immediate reward r_1 and a new state s_2 after the interaction with the environment. Analogously to the state- and state–action-values, we can define the value of an event as the expected discounted total reward of the on-line process $s_1, s_1^d, r_1, \ldots, s_t, s_t^d, r_t, \ldots$ starting from (x, y^d):

$$E^{\pi}(x, y^d) = \mathbf{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) | s_0 = x, s_1^d = y^d\right]. \tag{9}$$

Note that we should write E^{π_E, π_A}. However, according to (8), π_E and π_A determine a policy π, so the notation E^{π} is kept. Using (8) and (9), the event value function can also be expressed in terms of V:

$$E^{\pi}(x, y^d) = \sum_a \pi_A(x, y^d, a) \sum_y p(x, a, y)(r(x, a, y) + \gamma V^{\pi}(y)), \tag{10}$$

and conversely,

$$V^{\pi}(x) = \sum_{y^d} \pi_E(x, y^d)\, E^{\pi}(x, y^d). \tag{11}$$

From the last two equations the recursive formula:

$$E^{\pi}(x, y^d) = \sum_a \pi_A(x, y^d, a) \sum_y p(x, a, y)\left(r(x, a, y) + \gamma \sum_{z^d} \pi_E(y, z^d)\, E^{\pi}(y, z^d)\right) \tag{12}$$

can be derived. Eq. (12) can be simplified con-

[3] C{H} denotes the cardinality of the set H.

siderably. Denote by $p(y|x, y^d)$ the probability that given the initial state x and goal state y^d, the controller and the environment drive the system to state y in one step. Clearly,

$$p(y|x, y^d) = \sum_a \pi_A(x, y^d, a)\, p(x, a, y). \quad (13)$$

Furthermore, denote by $r(x, y^d)$ the expected immediate reward for the event (x, y^d), i.e.,

$$r(x, y^d) = \sum_a \sum_y \pi_A(x, y^d, a)\, p(x, a, y)\, r(x, a, y). \quad (14)$$

Using these notations, Eq. (12) can be written in the following form:

$$E^\pi(x, y^d) = r(x, y^d) + \gamma \sum_y p(y|x, y^d)\left(\sum_{z^d} \pi_E(y, z^d)\, E^\pi(y, z^d)\right). \quad (15)$$

Note that in an on-line process $s_1, s_1^d, r_1, \ldots, s_t, s_t^d, r_t, \ldots$ the state s_{t+1} is sampled from distribution $p(.|s_t, s_t^d)$, thus, the following SARSA-like value approximation can be used:

$$E_{t+1}(s_t, s_{t+1}^d) = (1-\alpha_t)\, E_t(s_t, s_{t+1}^d) + \alpha_t(r_t + \gamma E_t(s_{t+1}, s_{t+2}^d)). \quad (16)$$

The resulting algorithm is shown in Table 1.

We would like to emphasize that the value of event (x, y^d) depends (implicitly) on the controller. For example, in state x the value of y^d may be high, but if the controller is unable to get there, then the value of *trying* to get to y^d, i.e. $E(x, y^d)$, will be low.

As a consequence, it suffices to store event-values only for events (x, y) such that state y is 'close' to x (it is *achievable in one step* from x), and thus savings in storage space are possible.

To complete the algorithm, we have to specify a *controller* $\pi_A(x, y^d, a)$. This is the topic of the next section.

Table 1
Pseudo-code for the event learning algorithm

Initialize
$t := 0$;
$s_0 :=$ arbitrary, $r_0 := 0$;
Main loop
repeat
Observe s_t, r_t;
  Select s_{t+1}^d ε-greedily w.r.t. $E_t(.,.)$;
  Select a_t according to distribution $\pi_A(s_t, s_{t+1}^d, .)$
  take action a_t;
  % *Update*
  $E_{t+1}(s_{t-1}, s_t^d) := (1-\alpha_t)\, E_t(s_{t-1}, s_t^d) + \alpha_t(r_t + \gamma E_t(s_t, s_{t+1}^d))$;
  $t := t+1$;
End of loop

4. Robust policy heuristics

As was mentioned in Section 1, the time and state description of many real-life problems are continuous. We will use bold letters to emphasize that the appropriate variables are vectors of real values. Moreover, the continuous controllers in such problems usually operate on a desired *velocity* (v^d) instead of a desired state. However, event-learning requires controllers which operate on discrete time and state space and use state-desired state pairs instead of state-desired velocity pairs.[4] For the sake of simplicity, we assume that time is discretized uniformly into Δt intervals. As it is well known, for small Δt, $\frac{\mathbf{x}(t+\Delta t) - \mathbf{x}(t)}{\Delta t} \approx \dot{\mathbf{x}}(t)$, so $\frac{\mathbf{y}^d(t+\Delta t) - \mathbf{x}(t)}{\Delta t} \approx v^d(t)$. Thus, selecting a desired state in the discretized system can be accomplished by selecting a desired velocity in the continuous one:

$$\pi_A(x, y^d, a) := \tilde{\pi}_A\left(x, \frac{y^d - x}{\Delta t}, a\right), \quad (17)$$

where $\tilde{\pi}_A$ is the controller of the continuous system, i.e. $\tilde{\pi}_A(\mathbf{x}, \mathbf{v}^d, \mathbf{a})$ is the probability of selecting action $\mathbf{a}$ in state $\mathbf{x}$, when the desired velocity is $\mathbf{v}^d$.

4.1. Continuous dynamical systems

$\mathbf{x}$: $R \to D \subset R^n$ is a (first order) continuous dynamical system (CDS), if $\mathbf{x}(t)$ is continuously differentiable w.r.t. t, its derivative denoted by $\dot{\mathbf{x}}(t)$, and it satisfies the differential equation:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) \quad (18)$$

with some continuous f. (The possibility that the system could be controlled is not mentioned here.)

[4] It is assumed implicitly that the discretization is well conditioned, i.e. we are not concerned with the validity of the discretization.

From now on, dependence on t will not be explicitly denoted.

The dynamics of a physical system can typically be described by an equation of type (18). As a consequence, many important real-life problems can be modelled by continuous dynamical systems. Below we examine the controllability of such a system.

We assume that the dynamics of the system is given by[5]

$$\dot{\mathbf{x}} = \mathbf{P}(\mathbf{x})\,\mathbf{a} + \mathbf{q}(\mathbf{x}), \tag{19}$$

where $\mathbf{x} \in D \subset R^n$ is called the state vector of the system, $\mathbf{a} \in R^m$ is the control signal, and the continuous mappings $\mathbf{q}(\mathbf{x}) \in R^n$ and $\mathbf{P}(\mathbf{x}) \in R^{n\times m}$ characterize the dynamics of the system. We assume that D is compact and simply connected, and $\mathbf{P}(\mathbf{x})$ is invertible in the generalized sense, i.e. there exists a matrix $\mathbf{A}(\mathbf{x})$ for which $\mathbf{P}(\mathbf{x})\,\mathbf{A}(\mathbf{x})\,\mathbf{P}(\mathbf{x}) = \mathbf{P}(\mathbf{x})$. We also assume that both matrix fields $\mathbf{P}(\mathbf{x})$ and $\mathbf{A}(\mathbf{x})$ are differentiable w.r.t. $\mathbf{x}$.

4.2. The SDS controller

For the CDS described in Eq. (19), the inverse dynamics[6] is given by

$$\Phi(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{A}(\mathbf{x})\,\dot{\mathbf{x}} + \mathbf{b}(\mathbf{x}), \tag{20}$$

where $\mathbf{A}(\mathbf{x})$ is the (generalized) inverse of $\mathbf{P}(\mathbf{x})$, and $\mathbf{b}(\mathbf{x}) = -\mathbf{A}(\mathbf{x})\mathbf{q}(\mathbf{x})$. The inverse dynamics solves the control problem: $\Phi(\mathbf{x}, v^d)$ gives the control action that realizes desired velocity v^d in state $\mathbf{x}$.

However, finding Φ is usually a difficult (often intractable) problem, so we would like to use an easy-to-compute approximation $\hat{\Phi}$ instead. We assume that the approximate inverse dynamics has the form

$$\hat{\Phi}(\mathbf{x}, \dot{\mathbf{x}}) = \hat{\mathbf{A}}(\mathbf{x})\,\dot{\mathbf{x}} + \hat{\mathbf{b}}(\mathbf{x}). \tag{21}$$

The approximate inverse dynamics is allowed to be very coarse, since it can be corrected by an error term defined below. The resulting controller is called *SDS* controller:[7]

$$\mathbf{a}_{\mathrm{SDS}}(\mathbf{x}, \mathbf{v}^d) := \hat{\Phi}(\mathbf{x}, \mathbf{v}^d) + \Lambda \int_0^t \mathbf{w}(\tau)\,\mathrm{d}\tau, \tag{22}$$

where

$$\mathbf{w}(\tau) = \hat{\Phi}(\mathbf{x}(\tau), \mathbf{v}^d(\tau)) - \hat{\Phi}(\mathbf{x}(\tau), \dot{\mathbf{x}}(\tau)) \tag{23}$$

is the correction term, and $\Lambda > 0$ is the *amplification* or *gain* of the feedback.

The most important property of the SDS controller is that it can neglect the effects of the perturbation of the (inverse) dynamics (Szepesvári & Lörincz, 1996) (e.g. noise). For this reason, it can be called *robust* (Isidori, 1989).

Note that SDS is a deterministic controller, i.e. the distribution of the action values is given by

$$\tilde{\pi}_A^{\mathrm{SDS}}(\mathbf{x}, \mathbf{v}^d, \mathbf{a}) = \begin{cases} 1 & \text{if } \mathbf{a} = \mathbf{a}_{\mathrm{SDS}}(\mathbf{x}, \mathbf{v}^d), \\ 0 & \text{otherwise.} \end{cases} \tag{24}$$

The obtained controller can be easily inserted into the general event-learning scheme.

According to the SDS theory (see Theorem 2 in Appendix A and (Szepesvári, Cimmer, & Lörincz, 1997; Szepesvári & Lörincz, 1996)), only qualitative properness is required for $\hat{\Phi}$ (cf. the *sign-properness* condition). In general, such an approximation is easy to construct, e.g. simply by exploration: The $(\mathbf{x}, \mathbf{a}, \dot{\mathbf{x}})$ triplets can be tabulated for several $\mathbf{x}$, $\mathbf{a}$ pairs, and then this table can be used to search (truncate to or interpolate between) control $\mathbf{a}$ according to a given pair $(\mathbf{x}, v^d)$.

[5] It is well known that although the given dynamical system is of first order, this is not a real restriction, because dynamics of any order can be rewritten in this form (by extending the state space with the higher order derivatives $\dot{x}, \ddot{x}, \ldots, x^{(n)}$).

[6] Note that the inverse dynamics is not necessarily unique: $\Phi'(\mathbf{x}, \dot{\mathbf{x}}) = \Phi(\mathbf{x}, \dot{\mathbf{x}}) + (\mathbf{I} - \mathbf{A}(\mathbf{x})\,\mathbf{P}(\mathbf{x}))\,\mathbf{y}(\mathbf{x}, t)$ is also a valid inverse dynamics for arbitrary $\mathbf{y}(.,.)$. Learning of the inverse dynamics may be performed as suggested in (Fomin et al., 1997).

[7] A short description of the SDS control scheme is given in Appendix A. Detailed description of the motivation and properties of the SDS controller can be found in (Szepesvári et al., 1997; Szepesvári & Lörincz, 1996; Lörincz et al., 2001) and in our technical report (Lörincz et al., 2002).

4.3. Robust policy heuristics: applying SDS to event-learning

Theorem 2 ensures stability only for continuous systems and fixed $\mathbf{v}^d = \mathbf{v}(\mathbf{x})$. When it is applied to our event-learning algorithm, neither condition holds: the system is discretized and the desired velocity $\mathbf{v}^d$ is determined by policy π_E, which depends on E_t, so it varies with time.

Though, we may expect that the controller

$$\pi_A^{\mathrm{SDS}}(\mathbf{x}, \mathbf{y}^d, \mathbf{a}) := \tilde{\pi}_A^{\mathrm{SDS}}(\mathbf{x}, \mathbf{v}^d, \mathbf{a}) \tag{25}$$

preserves the stability and robustness of the original SDS controller. The resulting controller is called Robust Policy Heuristics. The algorithm of event-learning with Robust Policy Heuristics is shown in Table 2.

Note that in the discretized case, the output of the controller is the integral of the correction term in Eq. (22), which can be integrated easily[8] but the update occurs only at the start of a new event. Note also that the controller provides continuous (non-discrete) output, but the approximate inverse dynamics may still have a finite action set.

In the Appendices we show that the learning of the event-value function and the learning of the inverse dynamics can be accomplished simultaneously under the condition that the perfect inverse dynamics can be learnt. This is not the general case, however, and a stronger theorem with approximate learning of the inverse dynamics would be desired. Finally, we note that RPH is non-Markovian, because the controller relies heavily on history.

The next section presents computer simulations that demonstrate the specific properties of event-learning versus another (the SARSA) RL algorithm.

Table 2
Pseudo-code for the event-learning algorithm with RPH

Initialize
$i := 0$;
$s_0 :=$ arbitrary, $r_0 := 0$;
int_w $:= 0$;
Main loop
repeat
  Observe s_i, r_i;
  Select s_{i+1}^d ε-greedily w.r.t. E_i;
  $a_i := \hat{\Phi}(s_i, s_{i+1}^d) + \Lambda \cdot$ int_w;
  take action a_i;
  % *Update*
  int_w $:=$ int_w $+ \Delta t(\hat{\Phi}(s_{i-1}, s_i^d) - \hat{\Phi}(s_{i-1}, s_i))$;
  $E_{i+1}(s_{i-1}, s_i^d) := (1 - \alpha_i)\, E_i(s_{i-1}, s_i^d) + \alpha_i(r_i + \gamma E_i(s_i, s_{i+1}^d))$;
  $i := i + 1$;
end of loop

[8] In the interval $[t, t + \Delta t]$ the change is $\int_t^{t+\Delta t} \mathbf{w}(\tau)\, \mathrm{d}\tau = \mathbf{w}(t)\, \Delta t$, because $\mathbf{w}$ remains constant in this interval ($\dot{\mathbf{x}}$ is also measured at discrete time steps only).

5. Computational demonstrations

5.1. The rotational inverted pendulum

For the computer simulations the two-segment pendulum problem (Aamodt, 1997) was used. The pendulum is shown in Fig. 1. It has two links, a horizontal one (horizontal angle is α_1), a coupled vertical one (vertical angle is α_2) and a motor that is able to rotate both directions. The state of the pendulum is given by α_1, α_2, $\dot{\alpha}_1$ and $\dot{\alpha}_2$. For the equations of the dynamics see Appendix B. This particular task was chosen to demonstrate the strength of our method in solving real-world problems with noisy state-quantization continuous action spaces, and severely perturbed dynamics.

The task of the learning agent was to bring up the second link into its unstable equilibrium state and balance it there. To this end, it can express torque on the pendulum by using the motor. State variables were perturbed by small noise and were discretized (see Table 3). The controller 'sensed' only the code of the discretized state space. Discretization was uneven, a finer discretization was used around the bottom and the top positions of the vertical link. Different discretizations were utilized, one example can be seen in Fig. 2, where the division lines correspond to the borders of discretized states. The angle and the angular momentum of the horizontal link (see Fig. 1) are relatively less important from the point of view of motion and optimization. Dependence of the value function versus these variables is weak and is not shown in the figure.

The controller had two actions: $(-1.5,\ +1.5)$ in Newton meter [N m] units. An episode was considered successful if the pendulum was kept around its

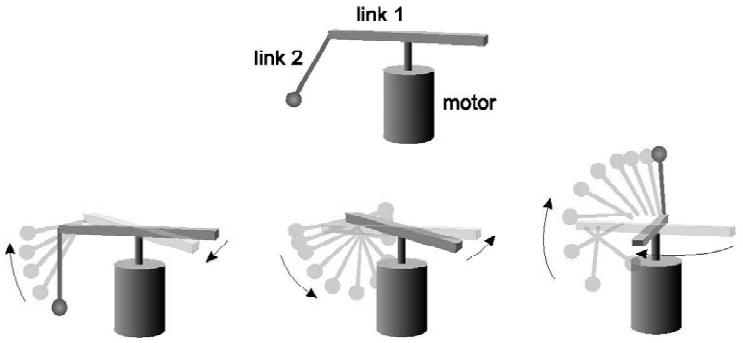

Fig. 1. The rotational inverted pendulum. Upper subfigure: the pendulum; lower subfigures: a successful episode shown in three consecutive series.

Table 3
Parameters of the computer simulations

Name of parameter	Value	Notation
SDS feedback gain	0.5–4	Λ
Resolution of the discretization of $\dot{\alpha}_2$	18 (5–21) parts	
Number of control actions	2	
Control actions (without SDS)	± 1.5 N m	
Average frequency of random control action	2 Hz	
Mass of horizontal link	0.82 kg	m_1
Mass of vertical link	0.43 kg	m_2
Length of horizontal link	0.35 m	l_1
Length of vertical link	0.3 m	l_2
Friction	0.005	K^{fricct}
Prescribed Standing Time	25 s	
Eligibility parameter	0.95	
Discount factor	0.98	γ
Learning rate	0.01	α

top position for a prescribed standing time (25 s). In all of the experiments the pendulum was started from its lowest position with zero angular velocities. A reward of 0 was given in each time step if angle α_2 was in $\pm 10^\circ$ proximity of π, otherwise penalty -1 was incurred.

For details about the applied approximate inverse dynamics see Appendix B. Technically, we have tabulated state–action–velocity triplets.[9] However, to save space, for a discretized state and action we have stored only the average of the resulting velocities (also computed in the discretized state space). The table thus needed $\mathcal{O}(|\mathcal{S}|\cdot|\mathcal{A}|)$ entries. To get $\hat{\Phi}(\mathbf{x}, \mathbf{v}^d)$, we have searched an entry $(\mathbf{x}, \bar{\mathbf{v}}_i, a_i)$ with minimum $\|\mathbf{v}^d - \bar{\mathbf{v}}_i\|$, and returned a_i. If no such entry was found, then a random action was returned. The obtained inverse dynamics is very coarse, and it may even violate the sign-properness condition during learning. The tabulation was made in parallel with the learning. This did not corrupt the results, since a good approximation was built up fast (much faster than the learning proceeded). This event selection method implies that since the desired event could only be a member of the set $\{(\mathbf{x}, \bar{\mathbf{v}}_i), i = 1, \ldots, |\mathcal{A}|\}$, only $|\mathcal{A}|$ event values were stored for each state. Thus, the total storage space needed by the algorithm was also $\mathcal{O}(|\mathcal{S}|\cdot|\mathcal{A}|)$.

The performance of our event-learning + RPH

[9] It was shown in Section 4 that $\mathbf{v}^d$ can be used instead of $\mathbf{y}^d$. This was utilized in the implementation of event selection as well.

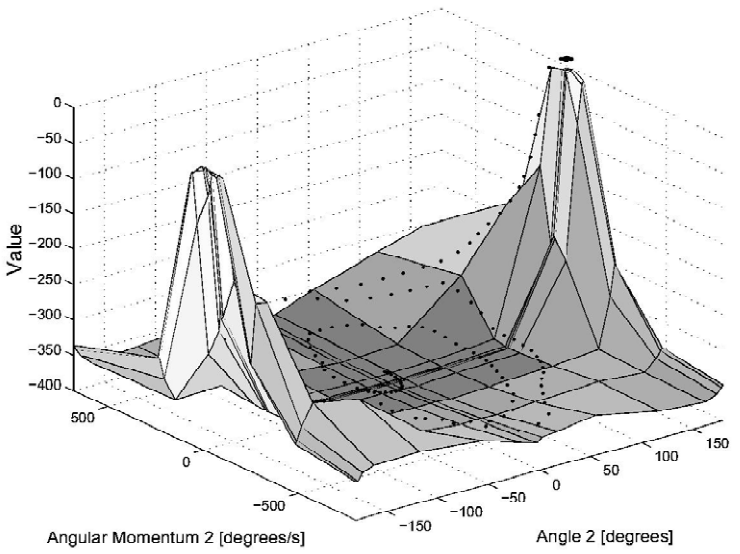

Fig. 2. Value function and an optimized trajectory. The vertical angle and vertical angular momentum are shown. (The value function depends to a much smaller extent on the horizontal angle and the horizontal angular momentum.) Lines signal the borders of the discretized domains of the state space. Values are gray level coded. Lighter colors denote higher values. A trajectory starting from the lowest position with zero velocity is depicted by dots.

algorithm was compared to the performance of SARSA. Eligibility traces were used in both algorithms to accelerate learning. The same parameters (learning rate, resolution of discretization, reward, eligibility decay, discount factor) were used for both algorithms. The parameters were taken from (Aamodt, 1997), and can be considered (near) optimal for the SARSA implementation (which was also taken from there). It should be noted, however, that event learning—via the backing SDS controller—has a coarse model. The main point of the 'comparisons' is that SARSA cannot cope with many real-world tasks because it assumes a discrete action space. In addition, increasing the resolution of that action space is problematic, because it increases the size of the table of Q-values that need to be learned—unlike in event-learning using RPH. Note also, that turning RPH off, the event-learning algorithm assumes a SARSA-like form (see Eq. (16)).

5.2. Experiments

5.2.1. The event-value function

Our algorithm was allowed to learn for 10,000 episodes to make sure that the value function does not change significantly any more, and it becomes near-optimal (in fact, a few hundred episodes would have been satisfactory).

The obtained value function cannot be graphed directly, since a state of the pendulum is a four-dimensional vector, so we would need $4+4+1$ dimensions to plot the event-value function. Thus, we plotted $V(x)$ instead of $E(x,y)$ with the two most significant state coordinates, α_2 and $\dot{\alpha}_2$ only. Values along the other two coordinates have been averaged.

The resulting value function is shown in Fig. 2. The trajectory of a successful trial is also depicted in the figure. It can be seen that the agent is not able to 'climb up' directly to the peak of the value function, but it has to reach it on a spiral trajectory instead. (Note that this spiral is closely related to the eigen-oscillation of the pendulum with increasing amplitude: the weaker the controller the more pronounced the connection.)

5.2.2. Comparison with SARSA

In this experiment we compared the learning speed of our algorithm and SARSA. We tested the two methods on the basic task: swinging up the pendulum to the the upper equilibrium state. Note, that

Table 4
Control actions available in the simulations[a]

Number of control actions					
2	±1.5				
4	±0.5	±1.5			
6	±0.5	±1.5	±2.5		
8	±0.5	±1.0	±1.5	±2.5	
10	±0.5	±1.0	±1.5	±2.0	±2.5
3	0	±1.5			

[a] Values are given in Newton meter [N m].

event-learning with RPH can select a continuous spectrum of actions (although based on two values), not just a few. To make this comparison 'fair', we tested SARSA also with 2, 3, 4, 6, 8 and 10 actions. The control action values are shown in Table 4.

Results are shown in Fig. 3. As it can be seen, event-learning + RPH learns much faster than SARSA. If SARSA has only two actions, it performs badly, because it can not approximate the true dynamics precisely enough. If it is allowed to use 10 actions, then the eventual performance is much improved, but learning is very slow because of the large size of the table of the Q-values. The figures demonstrate that contrary to SARSA, our algorithm is capable of fast learning and maintaining a good asymptotic performance (Note that SARSA variants with 3 to 10 actions asymptotically outperform event-learning.)

Event-learning with RPH unifies the advantages of these two extremities: the robust controller can achieve a good approximation and makes use of an approximate inverse dynamics with only two base actions, and that ensures small look-up table and fast learning. The 3-action SARSA performs particularly well, since it has the opportunity to do nothing, which proves to be very useful in this balancing task.

5.2.3. Distribution of completion time in a perturbed environment

The next set of simulations concerned the distribution of task execution time in a perturbed environment. Both algorithms were trained with $m_2 = 0.43$ kg. Before testing, we changed the mass to $m_2 = 0.53$ kg and tested RPH and SARSA on this new environment. No learning was allowed for this modified mass. Fig. 4 shows the results for this, imprecision', same for both types of computer runs. The points for the histogram were chosen in an equidistant manner. In turn, ticks refer to differing time intervals due to the log scale. It can be seen that in the case of SARSA, large deviations are possible, whereas our algorithm is able to control the system in a robust manner.

In the figure it can also be seen that strong

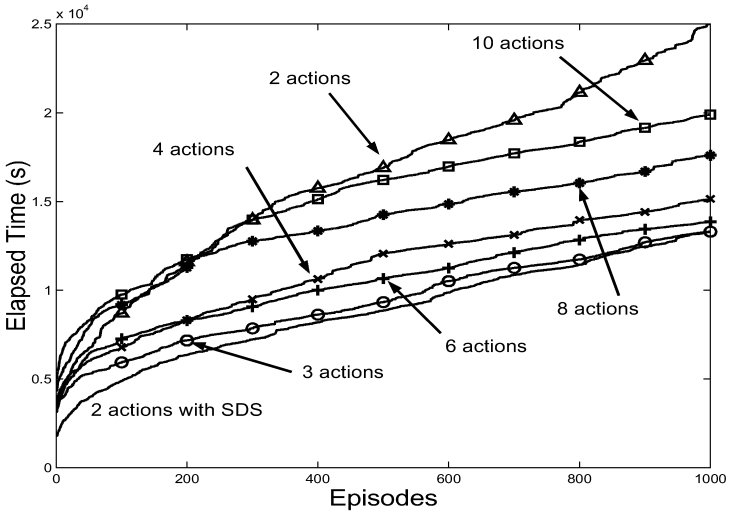

Fig. 3. Performance of SARSA and event-learning. The cumulative elapsed times for SARSA with 2, 3, 4, 6, 8 and 10 actions and event-learning. Smaller tangent slopes mean shorter episode times.

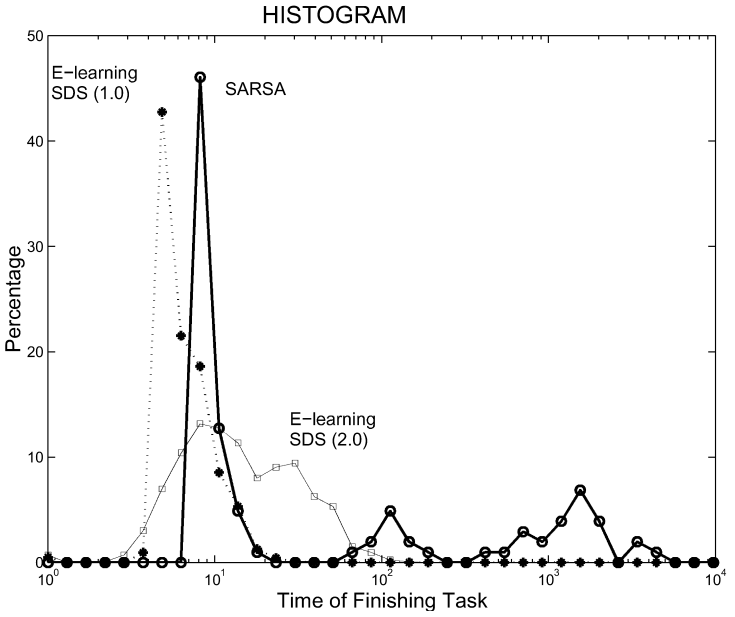

Fig. 4. Histograms of task completion time. Thick solid line: SARSA. Dotted line: event-learning with RPH ($\Lambda = 1.0$). Thin solid line: event-learning with RPH ($\Lambda = 2.0$).

feedback may also cause problems: $\Lambda = 1.0$ results are distinctly better than results for the $\Lambda = 2.0$ case. Coarse discretization which is attractive considering the need for infrequent decision making (see Section 5.2.5) can be limiting if perturbations are strong. Finer time resolution is required under this condition. Effects of delayed feedback[10] with the SDS controller has been studied elsewhere (Szepesvári & Lörincz, 1996).

5.2.4. Change of dynamics

Our most important question is: how large change in the dynamics can be handled by our method? To examine this, the two-segment pendulum was optimized for a given mass. In the optimization problem we used a coarse discretization in state space. After switching learning off, cases with different masses were tested. The described RPH method was used to compensate for the perturbation.

Fig. 5 depicts the results of the computer simulations. The figure shows the average task completion time for the two methods as a function of the mass change. The horizontal axis of the figure shows the change of the mass of the second link (in kilograms). With lighter (heavier) mass, the state–action policy finishes the task sooner (later). This is the straightforward consequence of the lack of robust control. Beyond about 0.1 kg (approx. a 25%) mass increase sharp deterioration takes place and performance of the state–action policy drops suddenly.

In contrast, event-learning with RPH starts to deteriorate only at around doubled mass. Small changes of the mass do not influence the task completion time significantly. One might say that the perturbations to the mass of one of the links are not as crucial as e.g. the length of the links—but both modify the optimal trajectory in state space. Theoretical considerations and computer simulations on this matter can be found in the literature (Lörincz, Hévizi, & Szepesvári, 2001). In turn, RPH may be suboptimal, but it can alleviate task execution and may not spoil optimization.

5.2.5. Frequency of decision making

As it was already mentioned, RPH makes decision

[10]Note that the term ‘delayed feedback’ concerns error compensation of the controller and has no connection to delayed rewards.

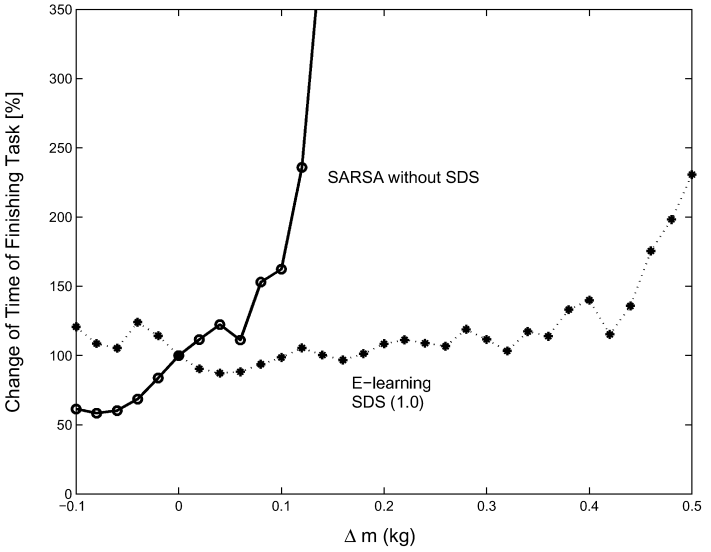

Fig. 5. Task completion time as a function of change of mass. Thick solid line: state–action controller pre-trained by SARSA and without RPH. Thin dotted line: event-learning with RPH ($\Lambda = 1$).

only if the system reaches a new state. It is obvious that eased constraints on frequent decisions can have advantages. The last figure depicts the average number of interactions with the system as a function of the number of discretization points (horizontal axis), i.e. the coarseness of the discretization. When discretization is too coarse then it corresponds to large and uneven perturbations. For fine discretizations the number of interactions grows because of the increased resolution. RPH is sensitive for delayed feedback (Szepesvári & Lörincz, 1996) and the number of interactions grows for very coarse discretization as well. There is a minimum in between, which is optimal if computational power or communication bandwidth are costly, and in turn, the minimization of interaction frequency is desired.

In contrast to event-learning + RPH, SARSA (and other traditional RL algorithms) has to make decisions on the next action in every time step. For 5 ms time step, this gives a 200 decision/s frequency, i.e. about 1000 decisions in an average 5 s long episode. On the other hand, event-learning + RPH results in a minimum interaction number which is about one fifth of the value that was achieved by SARSA (see Fig. 6). Note that this saving was achieved by optimizing only for a single state variable.

6. Conclusions and remarks

We have introduced the concept of event value function and its learning method, event-learning. The method supports the usage of a robust controller. The working domain of the optimized controller can be extended to strongly perturbed conditions. One disadvantage of the method is that the robust controller is sub-optimal. This fact may be compensated by the fast adaptation of the controller and the higher chance to achieve the goal in unexperienced situations.

It is important to note that learning time may increase exponentially with the number of degrees of freedom. In turn, it is essential to keep the number of degrees of freedom as low as possible, e.g. by considering a subset of state descriptors as noise, or by excluding some parameters from the optimization problem. The significance of robust controllers lies in the fact that they are able to deal with such

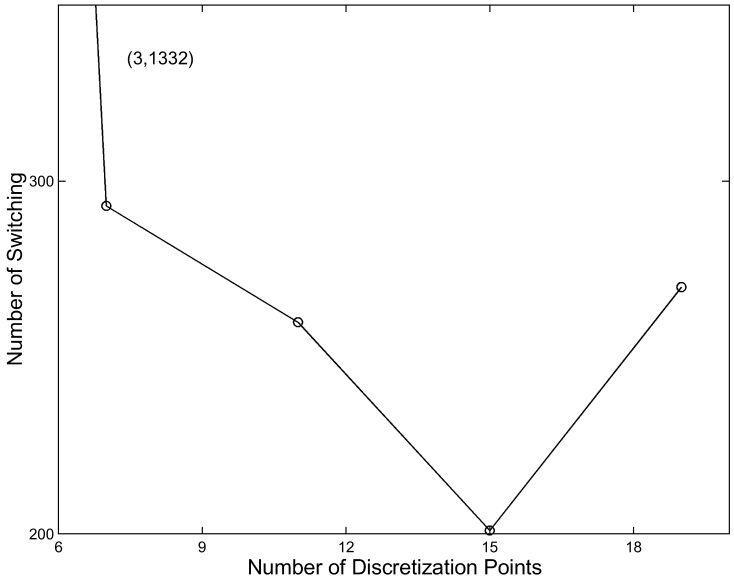

Fig. 6. Average number of interactions between the agent and the system. The interaction depends on the resolution in the case of event-learning with RPH. Low resolution gives rise to uncertainty about the actual position, motion is more erratic, and the number of interactions (i.e. the number of switches between events) increases. (The number of interactions was 1332 in the experiment when the number of discretization intervals was 3.) High resolution provides smooth motion, but the number of switches between modules increases. As a function of resolution there is a region where the number of interactions per episode is low.

situations, and thus ease the curse of dimensionality problem and accelerate learning. The gain in learning time may compensate the temporally sub-optimal nature of the actual solution.

We list a few properties of RPH and event-learning:

1. Storing the event-value function (when values are tabulated) requires theoretically $|\mathcal{S}|^2$ storage space. In practice, however, this can be reduced if $|\mathcal{A}| < |\mathcal{S}|$ (see Section 3 and Section 5.1). The theoretical lower bound on tabulation size is equal to $\min\{|\mathcal{S}|^2, |\mathcal{A}| \cdot |\mathcal{S}|\}$ for event-learning. This value may be significantly lower if not all state–state pairs can form predecessor–successor state–state pairs.
2. The robust controller is an attractive solution to reduce state space and, in turn, to reduce the search space: it is able to compensate for the coarse knowledge of state variables (or even the lack of information about some of them).
3. Learning of the event-value function requires neither the existence of an inverse dynamics, nor an underlying continuous dynamical system, it can be used together with a wide range of controllers, including state–action policies.
4. Event-learning can be written in analogous forms to value iteration methods like SARSA, and thus convergence theorems of RL can be established for this formulation as well.
5. Large savings may arise in the number of decision makings in the event-learning formulation. The main reason for this is that decision making is necessary only if the system reports that its state has changed.
6. Despite the fact that event-learning works with finite states, finite action space (continuously modified by RPH) and discrete time, RL control augmented with the robust controller extends the domain of applicability of RL to continuous tasks.

In the event-learning algorithm, the controller is given the task to drive the system from a state x to an other state y^d. This can be considered as setting a

subgoal. If the controller itself is also substituted by an RL algorithm, then a two-level hierarchical RL algorithm is obtained. The state description of the top level may be quite coarse, with a finer resolution on the second level. The potentials of this approach are promising, but have not been studied yet, and it is an open issue how to break a large problem into a hierarchical set of events. To our best knowledge, event-learning is unique from the point of view that it works with states that will (may) happen, and thus provides a natural framework for planning in modular hierarchical decision making problems.

Appendix A. The SDS control scheme

A.1. Robust speed field tracking

Finding an appropriate controller is closely related to the speed field tracking problem (SFT) (Hwang & Ahuja, 1992) described below:

Consider the continuous dynamical system defined in Section 4.1, and let $\mathbf{v}$: $D \to R^n$ be a prescribed *speed field*. Find a controller which can control the system so that if it gets to state $\mathbf{x}$, then the actual speed $\dot{\mathbf{x}}$ is close to the prescribed one ($\mathbf{v}(\mathbf{x})$). The task is called *robust*, if for a fixed $\varepsilon > 0$ $\|\mathbf{v}(\mathbf{x}(t)) - \dot{\mathbf{x}}(t)\| < \varepsilon$ for $t > 0$.

It is immediate that finding a robust controller for a given deterministic policy π_E is in fact a robust SFT problem. An analogous problem could be formulated for nondeterministic policies as well, but this will not be discussed here.

The planning abilities of SFT have been studied in the neurocontrol literature. Spreading activation type SFT algorithms have been designed (Lei, 1990; Keymeulen & Decuyper, 1992; Connolly & Grupen, 1993; Glausius, Komoda, & Gielen, 1995), and a unified framework has been provided (Szepesvári & Lörincz, 1998; Fomin, Rozgonyi, Szepesvári, & Lörincz, 1997). Planning and control can be unified in a neural (distributed connectionist) framework (Fomin et al., 1997) provided that SFT can be made robust against perturbations and changes in the system's dynamics.

As the below cited theorems show, the SDS controller provides a tractable solution to the robust SFT problem. However, in order to define this controller, several concepts need to be introduced.

A.2. Feedforward and feedback controllers

If planning and control are interleaved, i.e. at each time t the upgraded *instantaneous* (state) information is used to generate a new control signal, then the system will be called a *closed-loop system*. If the value of the control at time t depends only on the state of the plant at the same time, the control is said to be in a *static state feedback* control mode and the controller is called a *feedforward controller* (FFC). Assume that the planned motion and the actual motion are different. Then the difference, i.e. the error, can be used to generate an error-compensating signal. Generation of the error-compensating signal is the task of the *feedback controller* (FBC). (Note the ambiguous use of the term feedback.) The output of the feedback controller should be integrated in order to recall previous errors and thus to develop a *preventive compensatory control signal*. This means that a feedback controller applies *dynamic state feedback*, i.e. it is precisely the *dynamics* of the (compensatory) control signal that depends on the state of the plant as opposed to the case of static state feedback when the control signal itself depends on the state of the plant. In other words, in the case of dynamic state feedback the control signal is the output of another dynamical system. If, however, one views the problem from the aspect of the feedback controller, its output may depend only on the error, i.e. the feedback controller may itself be a feedforward control system working on the error as the state input. From this viewpoint the task of the feedback and that of the feedforward controller are similar: both should map *state values* to *control values*. In the following we use the term feedback control to refer to dynamic state feedback control.

Now, we again consider feedback control: the advantage of the extra FBC is that it allows the FFC to work with a broader range of problems since the FBC can compensate for errors. However, since feedback is working on the basis of a possible error, such an error first has to develop before any compensatory action can be made, i.e. feedback control is somewhat delayed.

The controller that realizes the approximate inverse dynamics plays a dual role: it computes the feedforward control signal that would move the system into the desired direction and, in case of error, the very same controller also computes the (feedback) compensatory signal. This compound controller will be called Static and Dynamic State Feedback Controller (SDS Feedback Controller). The equations describing the SDS controller with feedforward controller $\hat{\Phi}(\mathbf{x}, \mathbf{v}^d)$, feedback controller $\hat{\Phi}(\mathbf{x}, \mathbf{v}^d) - \hat{\Phi}(\mathbf{x}, \dot{\mathbf{x}})$ and feedback gain Λ are given in the article (Eq. (22)).

A.3. The stability theorem

To establish the theorem, we need the following definition.

Definition 1. Consider the autonomous system

$$\dot{\mathbf{x}} = f(\mathbf{x}), \tag{A.1}$$

where $\mathbf{x} \in D$, $D \subset R^n$ is compact, and f is a vector valued smooth function over D. The solution of Eq. (A.1) corresponding to the initial condition $\mathbf{x}(0) = \xi$, is denoted by $\phi(t;\ \xi)$, $(\xi \in D)$. Let the output of system (A.1) be

$$\mathbf{y} = h(\mathbf{x}),$$

where $\mathbf{y} \in R^m$, $m > 0$ integer, and h is continuous. Let $\|\cdot\|$ denote an arbitrary norm over R^m and let A be an arbitrary subset of R^m. We say that the output of the above system is *eventually uniformly bounded* (EUB) w.r.t. the set A if there is a bound $b > 0$ and a number $T > 0$ such that for each solution $\phi(t;\ \xi)$ for which $h(\xi) \in A$ it holds that $\|h(\phi(t;\ \xi))\| < b$ provided that $t > T$ and $\phi(t;\ \xi)$ is defined for t.

Now we are able to cite the stability theorem.

Theorem 2. *Let the CDS given by Eq. (19) be controlled by equation system (22). If*

1. $\hat{\Phi}(\mathbf{x}, \dot{\mathbf{x}}) = \hat{\mathbf{A}}(\mathbf{x})\,\dot{\mathbf{x}} + \hat{\mathbf{b}}(\mathbf{x})$,
2. $\mathbf{A}^{\mathrm{T}}(\mathbf{x})\,\mathbf{A}(\mathbf{x})$ *and* $\hat{\mathbf{A}}(\mathbf{x})^{\mathrm{T}}\,\mathbf{A}(\mathbf{x})$ *are uniformly positive definite over* D (*'sign-properness' condition*), *and*
3. *the speed field* $\mathbf{v}(\mathbf{x})$, $\hat{\Phi}(\mathbf{x}, \mathbf{v}(\mathbf{x}))$ *and* $\mathbf{A}(\mathbf{x})$ *are uniformly bounded and have uniformly bounded derivatives w.r.t.* $\mathbf{x}$ *over* D,

then for all $\Lambda > 0$ *the error of tracking speed field* $\mathbf{v}(\mathbf{x})$, $\mathbf{e}(\mathbf{x}) = \mathbf{v}(\mathbf{x}) - \dot{\mathbf{x}}$ *is EUB. The eventual bound* b *of the tracking error can be made arbitrarily small, more specifically* $b = \mathcal{O}(1/\Lambda)$ *and the necessary time for reaching* $\|\mathbf{e}\| \leqslant b$ *is proportional to* Λ.

The proof can be found e.g. in (Szepesvári & Lörincz, 1996). We note here that the joint formulation of reinforcement learning and CDS has been in the focus of research interest (Doya, 1996; ten Hagen & Kröse, 1998; Doya, 2000; ten Haagen, 2001). These works, however, do not use our event based formulation but treat the problem on different grounds.

Appendix B. The dynamics of the rotational inverted pendulum and the approximate inverse dynamics

B.1. The dynamics of the rotational inverted pendulum

Here we give the description of the CDS of the pendulum. We refer to the notations of Section 4.1. $\mathbf{x} = (\alpha_1,\ \alpha_2,\ \dot{\alpha}_1,\ \dot{\alpha}_2)$, thus, the state space is 4-dimensional, and $D = [-\pi,\ \pi] \times [-\pi,\ \pi] \times [-V_1,\ V_1] \times [-V_2,\ V_2]$, where V_1 and V_2 are physical bounds on the corresponding angular velocities. Control may be applied only to the angular acceleration of link 1.

Let the weight and length of link i be denoted by m_i and l_i, respectively $(i = 1, 2)$. Then the Lagrangian equation of motion is given by

$$\begin{bmatrix} K_1 + K_2 \sin^2 \alpha_2 & K_4 \cos \alpha_2 \\ K_4 \cos \alpha_2 & K_2 \end{bmatrix} \begin{bmatrix} \ddot{\alpha}_1 \\ \ddot{\alpha}_2 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} a \\ 0 \end{bmatrix}, \tag{B.1}$$

where a is the applied control, and

$$K_1 = (m_1 + m_2)l_1^2,$$

$$K_2 = m_2 l_2^2,$$

$$K_4 = m_2 l_1 l_2,$$

$$K_6 = m_2 l_2 g,$$

$$c_1 = -K_4 \dot{\alpha}_2^2 \sin\alpha_2 + K_2 \dot{\alpha}_1 \dot{\alpha}_2 \sin(2\alpha_2) + K_1^{\text{frict}} \dot{\alpha}_1,$$

$$c_2 = -1/2 K_2 \dot{\alpha}_1^2 \sin(2\alpha_2) + K_6 \sin\alpha_2 + K_2^{\text{frict}} \dot{\alpha}_2.$$

That is,

$$\mathbf{A}(\mathbf{x}) = \begin{bmatrix} K_1 + K_2 \sin^2\alpha_2 & K_4 \cos\alpha_2 \\ K_4 \cos\alpha_2 & K_2 \end{bmatrix}$$

and

$$\mathbf{b}(\mathbf{x}) = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$$

Consequently, $\mathbf{P}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x})$ and $\mathbf{q}(\mathbf{x}) = -\mathbf{A}^{-1}(\mathbf{x})\,\mathbf{b}(\mathbf{x})$ define the dynamics.

B.2. The approximate inverse dynamics

When the state space of the system is discretized, we have no access to $\mathbf{x}$, only to the discretized state $\hat{\mathbf{x}}$. In the case of the pendulum, the approximate inverse dynamics $\hat{\Phi}(\mathbf{x}, \mathbf{v}^d) = \Phi(\hat{\mathbf{x}}, \hat{\mathbf{v}}^d)$ satisfies the conditions of the SDS stability theorem.

Indeed, the only non-trivial condition is sign-properness, i.e. the positive definiteness of matrix $\hat{\mathbf{A}}^{\mathrm{T}}\mathbf{A}$ and $\mathbf{A}^{\mathrm{T}}\mathbf{A}$. To prove this, we show that both $\mathbf{A}$ and $\hat{\mathbf{A}}$ are positive definite.

The trace and the determinant of matrix $\mathbf{A}$ can be estimated from below as

$$\begin{aligned} K_1 + K_2 \sin^2\alpha_2 + K_2 &\geq K_1 + K_2 \\ &= m_1 l_1^2 + m_2 l_2^2 + m_2 l_2^2 \\ &> 0 \end{aligned} \tag{B.2}$$

and

$$\begin{aligned} \det A &= K_1 K_2 + K_2^2 \sin^2\alpha_2 - K_4^2 \cos^2\alpha_2 \\ &= m_1 m_2 l_1^2 l_2^2 + m_2^2 l_1^2 l_2^2 + m_2^2 l_2^4 \sin^2\alpha_2 \\ &\quad - m_2^2 l_1^2 l_2^2 \cos^2\alpha_2 \geqslant m_1 m_2 l_1^2 l_2^2 > 0, \end{aligned} \tag{B.3}$$

respectively, where $\sin^2\alpha_2$ was replaced by 0 and $\cos^2\alpha_2$ was replaced by 1. In turn this 2×2 matrix is positive definite. Approximating the true matrix $\mathbf{A}$ by matrix $\hat{\mathbf{A}} = \mathbf{A}(\hat{\mathbf{x}})$ at discretization point $\hat{\mathbf{x}}$ we have that $\hat{A}$ is uniformly positive for every discretization. In turn, the sign-properness conditions of the SDS controller are met.

Appendix C. Convergence of learning

In what follows, we relate our formulation to traditional formulations of RL. To simplify matters, we use the notation of Szepesvári and Littman (1999). For or notational simplicity, vector quantities will not be distinguished by bold letters in this section. The dynamic programming operator T that maps value functions to value functions can be written as

$$TV = \otimes \oplus (r + \gamma V), \tag{C.1}$$

where

$$\otimes = \max_a$$

and

$$\oplus = \sum_y p(x, a, y).$$

T has a unique fixed point, the optimal value function, denoted by V^*, for which $V^* = \otimes \oplus (r + \gamma V^*)$ holds (known as Bellman equation for V^*). The optimal value function is unique. This function determines the maximum of the expected value of the long term cumulated discounted immediate rewards that can be collected starting from any given state. The $\max_a$ operation selects the optimal action.

Now suppose that instead of action selection, a desired speed $v \in \mathrm{d}X$ is selected, where $\mathrm{d}X$ denotes differences between states. Computation of the difference may be executable, i.e. for discrete positions. In this case positions form a vector space and differences belong to the same vector space. Alternatively, the computation of the difference may not be executable, like in the case of tabulated states distinguished by indices. In both cases, the difference

will be called speed. Assume, that speeds are mapped to actions by the use of a controller $a = \hat{\Phi}(x, v)$. In our formalism this means that the action selection operator $\otimes$ is decomposed to a desired-speed selection operator $\otimes_E$ and a controller $\hat{\Phi}$. Consequently,

$$\begin{aligned} TV &= \otimes_E \oplus^{\Phi}(r + \gamma V) \\ &= \max_v \sum_y p(x, a \\ &= \hat{\Phi}(x, v), y)(r(x, a, y) + \gamma V(y)). \end{aligned}$$

Note that the distribution of the successor states depends on the controller.

Definition 3. Φ: $X \times \mathrm{d}X \to A$ is called the (*exact*) *inverse dynamics* of the system, if it maps the state pair (s, v) to action a so that taking action a in state s will move the system to state s' ($v = s' - s$). In this case we shall say that the exact inverse dynamics moves the system with speed v.

We define the speed value function $E^{\Phi} = \oplus^{\Phi}(r + \gamma V)$. For the optimal speed value function the following Bellman-type equation holds:

$$E^* = E^{\Phi} = \oplus^{\Phi}(r + \gamma \otimes_E E^*). \tag{C.2}$$

Note that from the viewpoint of speed value learning, the control selection is the part of the environment.

Speed is defined as the limit value of the difference between two states separated by small time interval divided by the duration of the time interval when this duration tends to zero. In this section speed will mean the difference between two states without making explicit reference to the time duration and the unit time. In turn, this formulation includes the traditional formulation. In this section the speed value function $E(x, v)(= E(x, y))$ will be used with $v = y - x$. One may make the distinction between speed value function $E(x, v)$ and event value function $E(x, y)$. For tabulated cases the two formulations are equivalent. We will need the following lemma.

Lemma 4. *Assume that the stochastic mappings* S_t: $Z \to W$ *and* R_t: $W \to X$ *converge to the* (*non-stochastic*) *mappings S and R, respectively, with probability one uniformly. If operators* R_t *are uniformly Lipschitz-continuous with constant K, then* $R_t \circ S_t \to R \circ S$ *with probability one uniformly.*

The proof of this lemma is left for the reader.

Consider the Q-learning type algorithm (Eq. (7)) in Fig. C.1.

set E_0, $\hat{\Phi}_0$ arbitrarily
repeat
  select v_{t+1} ε-greedily with respect to E_t
  $E_{t+1}(x_t, v_t) := (1 - \alpha_t)E_t(x_t, v_t)$
  $+ \alpha_t(r_t + \gamma E(x_{t+1}, v_{t+1}))$
  select action $a_{t+1} := \hat{\Phi}_t(x_{t+1}, v_{t+1})$
  select $\hat{\Phi}_{t+1}$
  take action a
  $t := t + 1$
end of loop

Fig. C.1. Learning algorithm for speed value function.

In this section, we will show that this algorithm converges to E^* under appropriate conditions.

Let the approximating operator sequence T_t be defined as follows:

$$T_t(E', E)(x, v) := \begin{cases} (1 - \alpha_t)E'(x, v) + \alpha_t(r_t + \gamma(\otimes_E E)(y)) & \text{if } (x, v) = (x_t, v_t), \\ E'(x, v) & \text{otherwise,} \end{cases} \tag{C.3}$$

where y is the state of the system after taking action $a = \hat{\Phi}_t(x, v)$ in state x.

Definition 5. T_t approximates T at E with probability one uniformly over X, if for $E_{t+1} = T_t(E_t, E)$ and arbitrary E_0, $E_t \to TE$ uniformly over X.

Lemma 6. T_t *approximates T at* E^* *with probability one uniformly over* $X \times \mathrm{d}X$, *if the following conditions hold*:

1. *the series of controllers* $\hat{\Phi}_t$ *converges to the exact inverse dynamics function with probability one uniformly over* $X \times \mathrm{d}X$,

2. *r_t has a finite variance and its expected value given x_t, a_t, y_t is equal to $r(x_t, a_t, y_t)$,*
3. *all possible (x, v) pairs are experienced infinitely often,*
4. *the coefficients $\alpha_t = \alpha_t(x_t, v_t)$ satisfy the Robbins–Monro conditions,*
5. *every $\hat{\Phi}_t$ is Lipschitz-continuous with constant K, i.e. there exists a constant K such that $\|\hat{\Phi}_t(x_1, v_1) - \hat{\Phi}_t(x_2, v_2)\| \leq K\|(x_1, v_1) - (x_2, v_2)\|$ for all x_i, v_i, t.*

Proof. $T_t = R_t S_t$, where $R_t = \hat{\Phi}_t$ and S_t is the Robbins–Monro-type iterated averaging. By condition (1), $R_t \to \Phi$ w.p.1 uniformly, by conditions (2)–(4) $S_t \to \oplus^{\Phi}(r + \gamma_E E)$. Consequently, by Lemma 4, $T_t(E)$ converges to $\oplus^{\Phi}(r + \gamma \otimes_E E)$ w.p.1 uniformly. □

Theorem 7. (Szepesvári & Littman, 1999) *Let T be an arbitrary mapping with fixed point V^*, and let T_t approximate T at V^* with probability one uniformly over X. Let V_0 be an arbitrary value function, and define $V_{t+1} = T_t(V_t, V_t)$. Let the set $\mathcal{V}(X)$ of bounded real-valued functions over X denote the set of value functions. If there exist 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 X*:

1. *for all U_1, $U_2 \in \mathcal{V}$ and all $x \in X$,*

$$|T_t(U_1, V^*)(x) - T_t(U_2, V^*)(x)| \leq G_t|U_1(x) - U_2(x)| \tag{C.4}$$

2. *for all $U, V \in \mathcal{V}$ and all $x \in X$,*

$$|T_t(U, V^*)(x) - T_t(U, V)(x)| \leq F_t \sup_{x'} |V^*(x') - V(x')| \tag{C.5}$$

3. *for all $k > 0$, $\prod_{t=k}^{n} G_t(x)$ converges to zero uniformly in x as n increases; and,*
4. *there exists $0 \leq \gamma < 1$ such that for all $x \in X$ and large enough t,*

$$F_t(x) \leq \gamma(1 - G_t(x)). \tag{C.6}$$

Corollary 8. *If the conditions of Lemma 6 hold, then the general speed learning algorithm converges to E^*.*

Proof. Let $X \times dX$ be the set of (x, v) pairs and $\mathcal{E}$ be the set of $X \times dX \to R$ speed value functions. Furthermore, let

$$G_t(x, v) = \begin{cases} 1 - \alpha_t, & \text{if } (x, v) = (x_t, v_t), \\ 1, & \text{otherwise} \end{cases} \tag{C.7}$$

and

$$F_t(x, v) = \begin{cases} \gamma\alpha_t, & \text{if } (x, v) = (x_t, v_t), \\ 0, & \text{otherwise.} \end{cases} \tag{C.8}$$

It is easy to check that T_t, G_t and F_t satisfy the conditions of Theorem 7. Therefore E_t converges to E^*. □

Remark 9. In words, the above theorem means that the learning of the speed value function and the learning of the inverse dynamics can be accomplished simultaneously.

From this result, the convergence of the SARSA-like learning algorithm with the following update rule follows, e.g. for ε-greedy policy (Singh et al., 2000):

$$E^{\pi}_{t+1}(x_t, v_t) = (1 - \alpha_t(x_t, v_t))\, E^{\pi}_t(x_t, v_t) + \alpha_t(x_t, v_t)(r_t + \gamma E^{\pi}_t(x_{t+1}, v_{t+1})). \tag{C.9}$$

It may be important to note that the convergence has been proved for a non-Markovian process.

Acknowledgements

Special thanks are due to Andy Barto for his detailed comments and suggestions made on an early version of this paper. Careful reading of the manuscript of one of our reviewers and helpful discussions with Csaba Szepesvári, Gábor Szirtes and Bálint Takács are gratefully acknowledged. This work was supported by the Hungarian National Science Foundation (Grant OTKA 32487). Thanks are due to Tor M. Aamodt (1997) for providing his rotational inverted pendulum software.

References

Aamodt, T. M. (1997). Intelligent control via reinforcement learning, Bachelor's thesis, University of Toronto, http://www.eecg.utoronto.ca/~aamodt/.

Andreae, J. (1969). Learning machines—a unified view. In Meetham, A. R., & Hudson, R. A. (Eds.), *Encyclopedia of information, linguistics and control*. Oxford: Pergamon Press, pp. 261–270.

Barto, A. (1978). Discrete and continuous models. *International Journal of General Systems*, *4*, 163–177.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Connolly, C. I., & Grupen, R. A. (1993). On the application of harmonic function to robotics. *Journal of Robotic Systems*, *10*, 931–946.

Doya, K. (1996). Temporal difference learning in continuous time and space. In *Advances in neural information processing systems*, vol. 8. Cambridge, MA: MIT Press.

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation*, *12*, 243–269.

Fomin, T., Rozgonyi, T., Szepesvári, C., & Lörincz, A. (1997). Self-organizing multi-resolution grid for motion planning and control. *International Journal of Neural Systems*, *7*, 757–776.

Glausius, R., Komoda, A., & Gielen, S. (1995). Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, *8*, 125–133.

Hwang, Y. K., & Ahuja, N. (1992). Gross motion planning – a survey. *ACM Computing Surveys*, *24*(3), 219–291.

Isidori, A. (1989). *Nonlinear control systems: an introduction*, 2nd ed. New York: Springer.

Jaakkola, T., Jordan, M. I., & Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, *6*(6), 1185–1201.

Keymeulen, D., & Decuyper, J. (1992). On the self-organizing properties of topological maps. In Varela, F. J., & Bourgine, P. (Eds.), Proceedings of the First European Conference on Artificial Life, *Toward a practice of autonomous systems*. Cambridge, MA: MIT Press.

Lei, G. (1990). A neural model with fluid properties for solving the labyrinthian puzzle. *Biological Cybernetics*, *64*, 61–67.

Littman, M. L., Cassandra, A., & Kaelbling, L. P. (1995). Learning policies for partially observable environments: scaling up. In *Proceedings of the 12th international conference on machine learning*. New York: Morgan Kaufmann.

Littman, M. L. (1996). Algorithms for sequential decision making, PhD thesis, Department of Computer Science, Brown University, February.

Lörincz, A., Pólik, I., & Szita, I. (2001). Event-learning and robust policy heuristics, Technical report NIPG-ELU-15-05-2001, ELTE, http://people.inf.elte.hu/lorincz/Files/NIPG-ELU-14-05-2001.pdf.

Lörincz, A., Hévizi, G., & Szepesvári, C. (2001). Ockham's razor modeling of the matrisome channels of the basal ganglia thalamocortical loops. *International Journal of Neural Computation*, *11*, 125–143.

Rummery, G. A., & Niranjan, M. (1996). *On-line Q-learning using connectionist systems*, Technical report. Cambridge University: Engineering Department.

Singh, S., Jaakkola, T., & Jordan, M. (1995). Learning without state-estimation in partially observable markovian decision processes. In *Proceedings of the 11th machine learning conference*, pp. 284–292.

Singh, S., Jaakkola, T., Littman, M. L., & Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, *38*, 287–303.

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

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, *8*, 1038–1044.

Szepesvári, C., & Lörincz, A. (1996). Approximate geometry representation and sensory fusion. *Neurocomputing*, *12*, 267–287.

Szepesvári, C., & Lörincz, A. (1998). An integrated architecture for motion-control and path-planning. *Journal of Robotic Systems*, *15*, 1–15.

Szepesvári, C., & Littman, M. L. (1999). A unified analysis of value-function-based reinforcement-learning algorithms. *Neural Computation*, *11*, 2017–2059.

Szepesvári, C., Cimmer, S., & Lörincz, A. (1997). Dynamic state feedback neurocontroller for compensatory control. *Neural Networks*, *10*, 1691–1708.

ten Haagen, S., & Kröse, B. (1998). Linear quadratic regulation using reinforcement learning. In *Proceedings of the 8th Belgian–Dutch conference on machine learning*.

ten Haagen, S. (2001). Continuous state space Q-learning for control of nonlinear systems, PhD thesis, University of Amsterdam, Amsterdam.

Tsitsiklis, J., & Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, *22*, 59–94.

Watkins, C. (1989). Learning from delayed rewards, PhD thesis, King's College, Cambridge, UK.

Witten, I. (1977). An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, *34*, 286–295.