The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

References Read

Important Assumptions

Important stuff

Control Delay in Reinforcement Learning for Real-Time Dynamic Systems: A Memoryless Approach

Erik Schuitema, Lucian Buşoniu, Robert Babuška and Pieter Jonker

***Abstract*— Robots controlled by Reinforcement Learning (RL) are still rare. A core challenge to the application of RL to robotic systems is to learn despite the existence of control delay – the delay between measuring a system's state and acting upon it. Control delay is always present in real systems. In this work, we present two novel temporal difference (TD) learning algorithms for problems with control delay. These algorithms improve learning performance by taking the control delay into account. We test our algorithms in a gridworld, where the delay is an integer multiple of the time step, as well as in the simulation of a robotic system, where the delay can have any value. In both tests, our proposed algorithms outperform classical TD learning algorithms, while maintaining low computational complexity.**

I. Introduction

Reinforcement Learning (RL) is a promising approach to adding autonomous learning capabilities to robotic systems. However, examples of real dynamic systems controlled in real-time by RL are still rare; most work on RL is done in simulation. An important difference between such real systems and their simulations is the presence of time delay between observation and control action: *control delay*. Every such real system that runs on-line RL will have a non-zero control delay due to transporting measurement data to the learning agent, computing the next action, and changing the state of the actuator. The delay is illustrated in Fig. 1. In this paper we show that besides negatively influencing the final solution, control delay can be particularly detrimental to the learning process itself, if it remains unaccounted for.

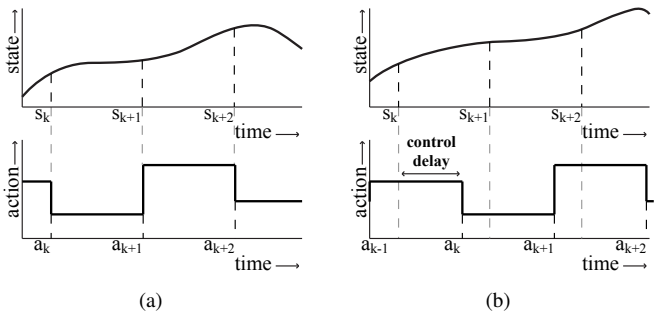

Fig. 1. Schematic illustration of control delay between measuring state s_k and acting accordingly with action a_k. a. No delay. b. With control delay.

E. Schuitema and P.P. Jonker are with the Delft Biorobotics Lab, and L. Buşoniu and R. Babuška are with the Delft Center for Systems and Control. Delft University of Technology, Mekelweg 2, 2628CD, Delft, The Netherlands. {e.schuitema, i.l.busoniu, r.babuska, p.p.jonker}@tudelft.nl

The influence of control delay on RL has received little attention in the RL literature. Currently, there are two state-of-the-art approaches. In the first approach, the state space of the learning agent is augmented with the actions that were executed during the delay interval [1]. While this approach works well, the state space increase can cause a large increase in learning time and memory requirements. The second approach is to learn a model of the underlying *undelayed* process, and use this model to base control actions on the future state after the delay, predicted by the model [2]. This adds the extra burden of acquiring a model of the system, while the added computational complexity may actually increase the delay itself.

As opposed to methods that augment the state space or estimate an explicit model, *memoryless* methods form an alternative approach. Such methods base the next control action only on the most recent observation. A memoryless method that is known empirically to produce good results is SARSA(λ) [3], [2]. The downside of memoryless approaches is that they are likely to perform suboptimally, because they have no means of predicting the state in which the control action will take effect. Furthermore, SARSA(λ) does not take the presence of delay into account in its learning updates. However, memoryless methods do not have the added complexity of learning a model or enlarging the state space, and may perform acceptably, especially when the delay is small.

In this work, we introduce two new memoryless, on-line algorithms – dSARSA(λ) being the most important one. While their complexity remains comparable to that of SARSA(λ), they exploit the knowledge about the length of the delay to improve their performance. In addition, we present an extension to these algorithms which is, under certain conditions, applicable to systems in which the delay is not an integer multiple of the time step. While this is most likely to be true for real robotic systems, this case has not been considered in previous literature on RL with delay.

This paper is organized as follows. In Sec. II, we discuss existing approaches to RL for systems with and without delay. In Sec. III, we present our main contribution. In Sec. IV, we compare our approaches to existing methods on two simulation problems, a gridworld problem and a robotic system. We end with conclusions in Sec. V.

II. Preliminaries

In this section, we define the Markov Decision Process framework for Reinforcement Learning, as well as an extension of this framework that models control delay.

A. The Markov Decision Process

The common approach in RL is to model the learning system as a Markov Decision Process (MDP) with discrete time steps labeled $k = 0, 1, .. \in \mathbb{Z}$ with sampling period h. The dynamic systems that we are interested in - robotic systems - will have a continuous state space S and a continuous action space A. The MDP is defined as the 4-tuple $\langle S, A, T, R \rangle$, where S is a set of states and A is a set of actions. The state transition probability function $T : S \times A \times S \rightarrow [0, \infty)$ defines the probability that the next state $s_{k+1} \in S$ belongs to a region $S_{k+1} \subset S$ as $\int_{S_{k+1}} T(s_k, a_k, s')ds'$, when executing action $a_k \in A$ in state $s_k \in S$. The reward function $R : S \times A \times S \rightarrow \mathbb{R}$ is real valued and defines the reward of a state transition as $r_{k+1} = R(s_k, a_k, s_{k+1})$. An MDP has the Markov property, which means that transitions only depend on the current state-action pair and not on past state-action pairs nor on information excluded from s.

The goal of the learner is to find an optimal control policy $\pi^* : S \rightarrow A$ that maps states to actions and that maximizes, from every initial state s_0, the return, i.e., the long-term sum of discounted rewards $\mathfrak{R}$:

$$\mathfrak{R}(s_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} \tag{1}$$

in which γ is the discount factor. The action-value function or Q-function $Q(s, a)$ gives the estimated return of choosing action a_k in state s_k and following the control policy afterwards: $Q(s_k, a_k) = \mathrm{E}\{r_{k+1} + \gamma \mathfrak{R}(s_{k+1})\}$.

Online RL implies that the system learns from interaction with the real world. The state transition probability function T can be either unknown to the learning agent (model-free RL), learned while learning the task (model-learning RL), or provided a priori (model-based RL). Although there are multiple classes of online RL algorithms, in this paper we will focus on the class of (online) Temporal Difference learning algorithms.

B. Temporal Difference learning

Temporal Difference (TD) learning methods mostly aim at directly estimating the Q-function $Q(s, a)$, from which the policy is derived by selecting the control action for which $Q(s, a)$ is maximal. Popular on-line algorithms in this category are *Q-learning* and *SARSA*. After every state transition, these algorithms update $Q(s_k, a_k)$ as follows:

$$\begin{array}{ll} Q(s_k, a_k) & \leftarrow Q(s_k, a_k) + \alpha \delta_{\mathrm{TD},k} \\ \delta_{\mathrm{TD_{SARSA}},k} & = r_{k+1} + \gamma Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k) \\ \delta_{\mathrm{TD_Q},k} & = r_{k+1} + \gamma \max_{a'} Q(s_{k+1}, a') - Q(s_k, a_k) \end{array} \tag{2}$$

where α is the learning rate, γ the discount factor and δ_{TD} the temporal difference (TD) error. While SARSA uses the Q-value of the actually selected action in its update rule, Q-learning uses the maximum achievable Q-value, which does not depend on the executed policy. This makes SARSA an *on-policy* and Q-learning an *off-policy* algorithm. In on-policy methods, $Q(s, a)$ is based on the policy that is used during action selection, while in off-policy methods, $Q(s, a)$ is based on a different (usually the optimal) policy, independent of the action selection policy.

To speed up convergence, SARSA and Q-learning can be combined with eligibility traces (see, e.g., [3]), thereby forming SARSA(λ) and Q(λ), respectively. With eligibility traces, learning updates are applied not just to the previously visited state-action pair (s_k, a_k), but also to pairs that were visited earlier in the episode. In this process, more recently visited (s, a)-pairs receive a stronger update than pairs visited longer ago. The update rules become

$$\begin{array}{ll} Q_{k+1}(s, a) & = Q_k(s, a) + \alpha \delta_{TD} e_k(s, a) \\ e_k(s, a) & = \begin{cases} 1 & \text{, if } s = s_k \text{ and } a = a_k \\ \gamma \lambda e_{k-1}(s, a) & \text{, otherwise.} \end{cases} \end{array} \tag{3}$$

which are now executed for all $s \in S, a \in A$. At the start of a new episode, e is reset to 0. For Q(λ), the eligibility of preceding states is only valid while the greedy policy is being followed. Thus, for Q(λ), e should also be reset after an exploratory action.

For Q-learning and SARSA, the greedy policy selects actions $a_{k,\mathrm{greedy}}$ according to

$$a_{k,\mathrm{greedy}} = \arg\max_{a'} Q(s_k, a') \tag{4}$$

which becomes a computationally costly operation when the action space is large (e.g. multidimensional). Therefore, these algorithms are likely to create delay in the online setting. The delay becomes larger when a computationally expensive function approximator is used to represent the Q-function.

A widely used action selection policy that includes exploratory actions is the ϵ-greedy policy, which selects actions $a_{k,\epsilon-\mathrm{greedy}}$ according to

$$a_{k,\epsilon-\mathrm{greedy}} = \begin{cases} a_{k,\mathrm{greedy}} & \text{, with prob. } 1 - \epsilon \\ random(A) & \text{, with prob. } \epsilon \end{cases} \tag{5}$$

C. Control delay in MDPs

We now consider MDPs with control delay. We define control delay as the time delay between the moment of observing the state and the moment when acting upon that state takes effect. Control delay, which we will refer to simply as delay, can be caused both by delayed observation, e.g., due to transportation of the measured data, and by delayed actuation, e.g., due to lengthy computations. In this paper, we only consider constant delays[1]. We define the relative delay τ_d as

$$\tau_d = \frac{T_d}{h} \tag{6}$$

with T_d the absolute delay and h the sampling period.

In [1], it is shown that from the point of view of the learning agent, there is no functional difference between observation delay and action delay; both add up to the delay between the moment of measurement and the actual action.

From (2), we can see that the Q-function is adjusted every time step according to a supposedly Markovian (stochastic)

[1]Variable delays in real robotic systems (interesting for future work) can be made (nearly) constant by adding artificial delay to 'fast samples'.

state transition based on state s_k and action a_k; the agent learns the effect of executing action a_k. In the delayed case, the action that is executed in s_k is not a_k. If τ_d is an integer, i.e., the delay is an integer multiple of h, the actually executed action is $a_{k-\tau_d}$. If τ_d is not an integer, a state transition is influenced by *two* actions selected in the past.

The fact that state transitions become dependent on actions selected in the past, which are not part of the input of the learning agent, results in a violation of the Markov property. This relates the problem of delay to the framework of Partially Observable MDPs, or POMDPs.

D. Existing approaches to MDPs with delay

Delay implies that decisions take effect in future states. When the future state (distribution) can be predicted from the most recent observation and the upcoming actions, e.g. by an explicit state transition model, optimal action selection becomes possible again. From [1], it is known that when the state space of the MDP is expanded with the actions taken in the past during the length of the delay, forming the augmented state space $I_{\tau_d} = S \times A^{\tau_d}$ with τ_d integer-valued, a constant delay MDP can be reduced to the regular MDP $\langle I_{\tau_d}, A, T, R \rangle$. This formulation makes it possible to use existing RL techniques to solve the delayed case. However, since the state space dimensionality grows with the number of delay steps, learning time and memory requirements will rapidly increase with this approach.

In [2], an approach called Model Based Simulation is presented, in which a state transition model of the underlying *undelayed* MDP is learned by matching actions with the states in which they actually took place. Model-based RL is then used to estimate the optimal (action-)value function. However, such an approach has the additional burden of learning an explicit model of the system.

A *memoryless policy* is a policy that only bases its actions on the current state s, despite the delay. This means that it does not take into account the future state in which the action takes effect. Therefore, it is likely to perform worse than methods that use a model for state prediction. From [4], it is known that the best memoryless policy of a POMDP can be arbitrarily suboptimal in the worst case. However, this does not mean that a memoryless policy cannot achieve an acceptable level of performance in a given problem. In [5], it is argued that SARSA(λ) performs very well in finding memoryless policies for POMDPs, compared to more sophisticated and computationally much more expensive methods. In [2], Model Based Simulation is also compared with SARSA(λ), which performs surprisingly well, but not as well as their model-based approach.

In our main contribution, we will use the knowledge on the source of the partial observability - the delay, and its length - to create memoryless algorithms that outperform SARSA(λ), while having similar complexity. They do not enlarge the state space and they are model-free.

In all the aforementioned work, only delays of an integer multiple of the time step were considered, while in real-time dynamic systems, this is usually not the case. Therefore, we also present an extension to our algorithms that make them, under certain conditions, applicable to the case where τ_d can have any non-integer value.

III. TD learning with control delay: dSARSA and dQ

We now present our main contribution: modified versions of SARSA and Q-learning that exploit knowledge about the delay. In these versions, instead of updating $Q(s_k, a_k)$, updates are performed for the *effective* action $\hat{a}_k$ that actually took place in s_k. We will first consider the case where τ_d is an integer, which results in the following effective action

$$\hat{a}_k = a_{k-\tau_d}. \tag{7}$$

The update rules are as follows

$$\begin{aligned} Q(s_k, \hat{a}_k) &\leftarrow Q(s_k, \hat{a}_k) + \alpha \delta_{\mathrm{TD},k} \\ \delta_{\mathrm{TD_{dSARSA},}k} &= r_{k+1} + \gamma Q(s_{k+1}, \hat{a}_{k+1}) - Q(s_k, \hat{a}_k) \\ \delta_{\mathrm{TD_{dQ},}k} &= r_{k+1} + \gamma \max_{a'} Q(s_{k+1}, a') - Q(s_k, \hat{a}_k) \end{aligned} \tag{8}$$

We call these variants dSARSA and dQ, where 'd' stands for 'delay'. Both algorithms are memoryless, which means their policy depends only on the current state. Action *execution* is still delayed. They use the knowledge on the length of the delay to improve the learning updates. Eligibility traces can be introduced by modifying (3) at the following point: $e_k(s, a) = 1$ if $s = s_k$ and $a = \hat{a}_k$. We will now discuss the most important properties of dQ and dSARSA.

A. Behavior of dQ-learning

Regular Q-learning is an off-policy algorithm, which means that $Q(s, a)$ is not based on the action selection policy. The only restriction on the action selection policy is that all state-action pairs continue to be updated. In dQ-learning, we restored the temporal match between states and actions. This means that with dQ-learning, the action-value function will converge to the optimal action-value function of the underlying *undelayed* MDP. Execution is still delayed.

When combining dQ-learning with eligibility traces, forming dQ(λ)-learning, convergence is not guaranteed, since eligibility of preceding states is only valid when the greedy policy is being followed. With delayed execution, this is generally not the case. In our empirical evaluations in Sec. IV, we will indeed see that the use of eligibility traces can lead to rapid divergence. However, dQ(0) is still an interesting algorithm due to its convergence properties, and we expect it to be an improvement over regular Q-learning.

B. Behavior of dSARSA

Regular SARSA is an on-policy algorithm. Therefore, the estimated Q-function is based on the policy's action selection probabilities. In the case of dSARSA, execution of actions is still delayed. Therefore, although we restored the temporal match between states and actions in the updates, the actual policy still depends on the history of the agent. The convergence proof of SARSA [6] requires the policy to be greedy in the limit with infinite exploration (GLIE), which

dSARSA cannot provide. dSARSA can choose, but not execute, greedy actions with respect to its value function due to the delayed execution of actions. This convergence proof therefore does not hold. However, we expect that dSARSA is still an improvement over SARSA due to the incorporation of knowledge about the delay. When combining dSARSA with eligibility traces, forming dSARSA(λ), the eligibility of preceding states is in this case justified because the action-value function is based on the actually executed policy (the delayed policy). Therefore, dSARSA and dSARSA(λ) are expected to give the same solution.

From the above reasoning and from existing convergence proofs, we can conclude that from the algorithms Q(λ), dQ(λ), SARSA(λ) and dSARSA(λ), only dQ(0) (i.e., without eligibility traces) is guaranteed to converge in the delayed case. More insight into the convergence of dSARSA(λ) and dQ(λ) remains important future work.

The actions that are selected prior to visiting a state s are responsible for the action actually executed in s. While these actions can in principle not be predicted from s, their probability distribution might contain structure when the policy is quasi-stationary and the distribution of initial states is fixed. When the learning rate α is reduced, the policy changes less rapidly, and Q-values represent the average of a larger number of experiences. From this reasoning, we expect the performance of memoryless policies to improve with decreasing α. In Sec. IV, we will verify this hypothesis.

Note that although dSARSA and dQ are memoryless and model-free, they can still benefit from a (learned) state transition model by using it to predict the future state after τ_d steps and selecting the action for that predicted state.

C. Non-integer values of τ_d

We will now consider the case where τ_d is not an integer. From Fig. 1, it can be seen that the control action is a combination of the two previously selected actions $a_{k-\lceil\tau_d\rceil}$ and $a_{k-\lceil\tau_d\rceil+1}$ with $\lceil\tau_d\rceil$ the smallest integer for which $\tau_d \leq \lceil\tau_d\rceil$. Therefore, dSARSA and dQ cannot be directly applied in this case. We will now show that in the special case where the system dynamics can be accurately *linearized* at arbitrary states, over the length of one time step, it is possible to estimate the effective action $\hat{a}(kh)$ *without* knowledge of the system dynamics. In case $T_d < h$, [2]the state transition of the real system can be shown to be, in a first approximation, equivalent to the case where the system would have executed the following *virtual* effective action $\hat{a}(kh)$ during the full time step:

$$\hat{a}_k = a_{k-1}\tau_d + a_k(1-\tau_d) \tag{9}$$

Here we assume that actions can be linearly combined, e.g., when actions are motor torque or voltage. Switching to the continuous time notation $s(kh) = s_k$, $a(kh) = a_k$, the locally linearized system at time $t = kh$ around state $s(kh)$ has the form

$$\dot{s}(t) = Fs(t) + Ga(t) \tag{10}$$

[2]To avoid notation clutter, we make the assumption that $T_d < h$. However, generalizing the results to larger delays is straightforward.

with F and G matrices. The exact solution to (10) in the delayed case with $T_d < h$ is

$$\begin{aligned} s(kh+h) &= e^{Fh}s(kh) \\ &+ \int_{kh}^{kh+T_d} e^{F(kh+h-s)}\,ds\,Ga(kh-h) \\ &+ \int_{kh+T_d}^{kh+h} e^{F(kh+h-s)}\,ds\,Ga(kh) \end{aligned} \tag{11}$$

The exact solution to the non-delayed case in which action $\hat{a}(kh)$ from (9) was executed is

$$\begin{aligned} s(kh+h) &= e^{Fh}s(kh) \\ &+ \int_{kh}^{kh+h} e^{F(kh+h-s)}\,ds\,Ga(kh-h)\frac{T_d}{h} \\ &+ \int_{kh}^{kh+h} e^{F(kh+h-s)}\,ds\,Ga(kh)\frac{h-T_d}{h} \end{aligned} \tag{12}$$

For small time steps, we can approximate integrals of the form $\int_{t_1}^{t_2} e^{Xs}\,ds \approx I(t_2-t_1) + \frac{1}{2}X(t_2^2-t_1^2) \approx I(t_2-t_1)$, for which (11) and (12) become equal. In this special case, dSARSA and dQ can again be applied using (9).

IV. Empirical evaluations

In this section, we empirically evaluate the performance of dSARSA and dQ on two test problems. We consider integer values of τ_d in the large W-maze – a gridworld problem – and non-integer values of τ_d in the simulation of the two-link manipulator – a robotic system.

A. Integer values of τ_d: the large W-maze

We first consider the large W-maze, see Fig. 2b, in which τ_d is always an integer. This is a modified version of the W-maze, see Fig. 2a, that was used in [2] to empirically illustrate several approaches to delayed MDPs. In the original W-maze, a delay of one time step easily results in the agent missing the middle corridor that leads towards the goal. The larger corridors and goal state of the large W-maze make the effect of delayed policy execution less severe.

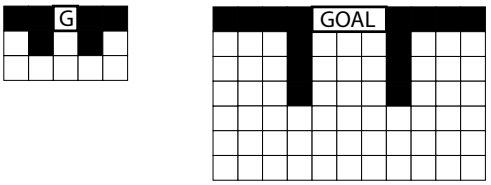

Fig. 2. The W-maze (left) and the large W-maze (right), which has larger corridors and a larger goal region.

We evaluated the performance of dQ(0), dQ(λ) and dSARSA(λ) in comparison with Q(λ) and SARSA(λ). While dSARSA(0) is merely a slower version of dSARSA(λ) and of less interest, dQ(0) is the only algorithm with convergence guarantees (see Sec. III-A) and we therefore include it. We

set the delay to 2 time steps. The only reward in the system is -1 for every action, so that the agent learns to reach the goal as fast as possible. We did not use discounting ($\gamma = 1$). We used the ϵ-greedy policy (5) with $\epsilon = 0.1$ (fixed).

First, we compared the policy performance from the solutions found by all methods, as a function of the learning rate α. While keeping α constant, the agent was allowed to learn for $1 \cdot 10^6$ time steps. The policy performance was periodically measured by letting the agent start a trial from each of the 60 possible positions in the maze, execute the greedy policy until the goal was reached (with a maximum of 300 steps), and summing up all trial times (ergo, smaller is better). The average of all performance measurements during the last $5 \cdot 10^5$ steps is plotted against α in Fig. 3.

The graph shows that for all methods, except dQ(0), the performance is much better for smaller values of α. This confirms the hypothesis that memoryless policies for delayed MDPs can improve their performance by averaging over more samples. This averaging is not needed by dQ(0). Despite its lack of eligibility traces, dQ(0) can still learn fast because it allows for high values of α.

Furthermore, we can observe that at equal values of α, dSARSA(0.8) performed at least twice as good as SARSA(0.8). Instability of dQ(0.8) occurred for larger values of α (Q-values quickly went to infinity).

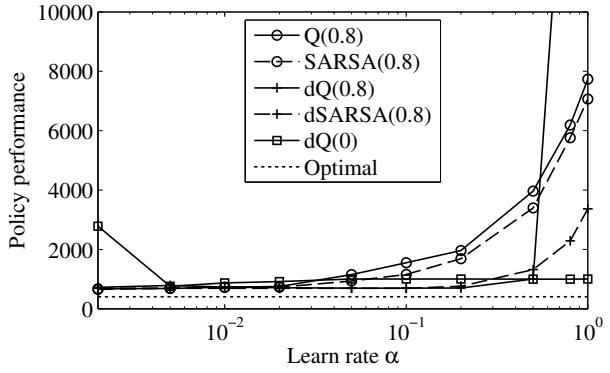

Fig. 3. Comparison of several memoryless TD algorithms on the large W-maze problem, showing the final average performance (lower is better) at different values of α, with α on a log scale (average of 20 runs).

To compare the learning speed of all methods, we again kept ϵ and α constant. However, we chose a different α for each method for the following reason. An increase in α results in faster learning, but also in a decrease in final performance. Therefore, we chose the largest values of α for which each method was able to achieve approximately the same final performance of 800 or better. This resulted in the following values: dQ(0): $\alpha = 1$, dQ(0.8): $\alpha = 0.2$, dSARSA(0.8): $\alpha = 0.02$, Q(0.8): $\alpha = 0.02$, SARSA(0.8): $\alpha = 0.2$. dQ(0) is an exception; for this method, the average performance did not drop below approx. 1000. The result can be found in Fig. 4. We can observe that in order to achieve the same final performance, dSARSA(0.8) learns much faster than SARSA(0.8) and Q(0.8). Although dQ(0) does not benefit from eligibility traces, in this particular test it can learn as fast as dQ(0.8) and dSARSA(0.8) because it allows for much higher values of α.

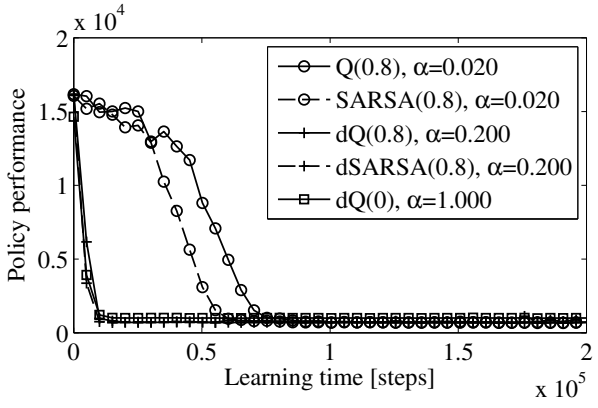

Fig. 4. Learning curves of several memoryless TD algorithms on the large W-maze problem. The learn rate α for each method was set to a value that produced a performance measure < 800 (average of 20 runs).

B. Non-integer values of τ_d: the two-link manipulator

We now consider the simulation of the two link manipulator - a robotic system - depicted in Fig. 5. This system has been used in previous work (see, for example, [7]), has limited complexity and is well reproducible. In this system, the delay can be any non-integer multiple of the time step h, which is generally the case for robotic systems. The system has two rigid links, which are connected by a motorized joint. One end of the system is attached to the world, also with a motorized joint. The system moves in the two dimensional horizontal plane without gravity according to the following fourth-order non-linear dynamics:

$$M(\boldsymbol{\varphi})\ddot{\boldsymbol{\varphi}} + C(\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})\dot{\boldsymbol{\varphi}} = \boldsymbol{\tau} \tag{13}$$

where $\boldsymbol{\varphi} = [\varphi_1, \varphi_2]$, $\boldsymbol{\tau} = [\tau_1, \tau_2]$, $M(\boldsymbol{\varphi})$ is the mass matrix and $C(\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}})$ is the Coriolis and centrifugal forces matrix.

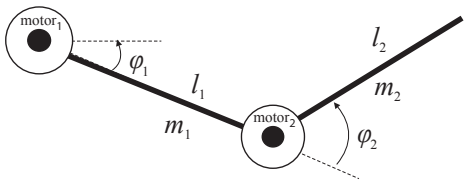

Fig. 5. Schematic overview of the two-link manipulator

We consider delay values of $0 <= T_d <= h$. The task of the system is to accomplish $\varphi_1 = \varphi_2 = \dot{\varphi}_1 = \dot{\varphi}_2 = 0$ as fast as possible by choosing torque signals for both motors. To this end, a reward is given when the angles and angular velocities are within a small region around 0: $|\varphi| < 0.17$ (rad) and $|\dot{\varphi}| < 0.2$ (rad) $\cdot\, \mathrm{s}^{-1}$. Furthermore, a time penalty is given at each time step. The reward function r is

$$r_t = \begin{cases} 100, & \text{if } |\varphi_t| < 0.17 \text{ and } |\dot{\varphi}_t| < 0.2 \\ -1, & \text{all other cases (time penalty)} \end{cases} \tag{14}$$

The learning time step $h = 0.05$s, the learning rate $\alpha = 0.4$, the exploration rate $\epsilon = 0.05$ (again using the ϵ-greedy

policy (5)), the discount factor $\gamma = 0.98$ and the trace decay rate $\lambda = 0.92$. We use tile coding function approximation [3] with 16 tilings to approximate the Q-function. The state $s = (\varphi_1, \varphi_2, \dot{\varphi}_1, \dot{\varphi}_2)$. The tile widths for the state space are 1/12 (rad) in the φ_1 and φ_2 dimensions and 1/6 (rad) $\cdot\, \mathrm{s}^{-1}$ in the $\dot{\varphi}_1$ and $\dot{\varphi}_2$ dimensions. The tile widths for the action space are 1Nm in both dimensions. The action space $A = \tau_1 \times \tau_2$ is divided in 5 equidistant discrete steps in both dimensions, allowing for a total of $5 \cdot 5 = 25$ torque combinations. These parameters were selected empirically because they produced stable learning; they were not systematically optimized.

We first tested the effects of a delay of one time step or less using regular SARSA(λ). The results can be found in Fig. 6. We can observe that for this particular system and for $\alpha = 0.4$, the system without delay smoothly converges. Delay values of $T_d \geq 0.025$s ($\tau_d \geq 0.5$), however, slow down the learning process or cause frequent divergence/unlearning. With a delay of a full time step, the system hardly learns for this value of α.

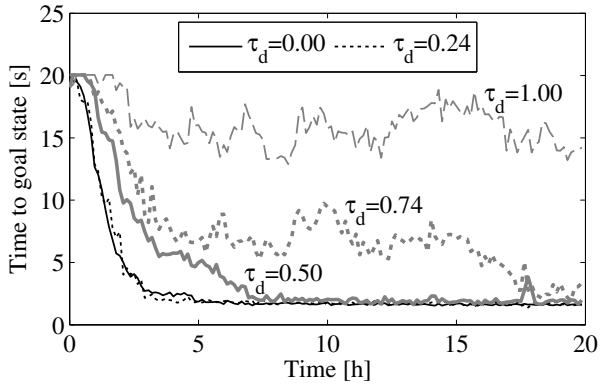

Fig. 6. Learning curve of the two-link manipulator running regular SARSA(λ) for $\tau_d = 0, 0.24, 0.50, 0.74$ and 1.0 (average of 20 runs).

Next, we tested dSARSA(λ), using (9) to calculate the effective action. To this end, we set the delay to $T_d = 0.037$s ($\tau_d = 0.74$), a value which caused convergence problems for regular SARSA(λ). We compared it against the approach of using augmented state space $I_{\tau_d} = S \times A^{\tau_d}$ with $\tau_d = 1$, so that $s_{aug} = (s, \tau_{1,k-1}, \tau_{2,k-1})$ (tile widths for $\tau_{1,k-1}, \tau_{2,k-1}$ are 1Nm). The results can be found in Fig. 7. We can observe that for both approaches, the delayed system nicely converges to a solution, unlike with SARSA(λ) (see Fig. 6). With dSARSA(λ), the system learns faster than with SARSA(λ) with augmented state space, but slower than the original, non-delayed system with SARSA(λ). Both methods have statistically indistinguishable final system performance. This suggests that the possible performance increase from a model based approach, such as in [2], will only be marginal in this case.

V. Conclusions

In this work, we derived new memoryless, model-free, online TD learning algorithms for MDPs with delay – dSARSA(λ) being the most important one – that perform

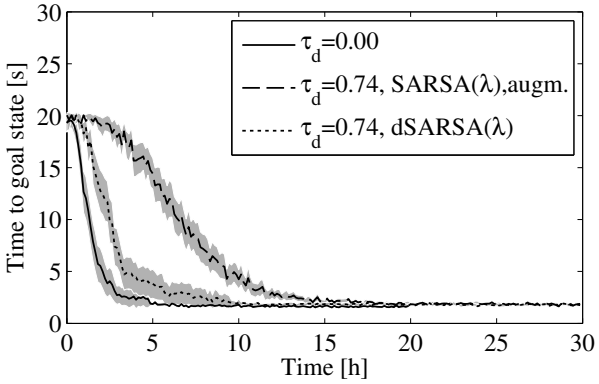

Fig. 7. Learning curves of the two-link manipulator with a delay of $\tau_d = 0.74$, comparing dSARSA(λ) and SARSA(λ) with augmented state space, showing the average of 60 runs. The shaded area shows the 95% confidence bounds on the mean.

better than classical TD algorithms by exploiting knowledge about the delay. We showed in a gridworld problem that dSARSA(λ) can significantly outperform Q(λ) and SARSA(λ) in terms of learning speed and final policy performance.

We extended our algorithms to systems in which the delay is not an integer multiple of the time step. We showed in a simulation of a robotic system that a control delay of less than a single time step can already degrade learning performance with classical SARSA(λ), while dSARSA(λ) can successfully learn in that case. dSARSA(λ) learned faster than a memory based approach – SARSA(λ) with augmented state space – while its policy performance did not degrade.

We can conclude that dSARSA(λ) is a better memoryless approach than classical SARSA(λ) for MDPs with delay, and that in some cases, it can compete with memory based approaches. In future work, we will test the applicability of our methods on our humanoid robot 'LEO' [8].

References

[1] K. V. Katsikopoulos and S. E. Engelbrecht, "Markov decision processes with delays and asynchronous cost collection," *IEEE Transactions on Automatic Control*, vol. 48, no. 4, pp. 568–574, 2003.

[2] T. J. Walsh, A. Nouri, L. Li, and M. L. Littman, "Learning and planning in environments with delayed feedback," *Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 1, pp. 83–105, 2009.

[3] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 1998.

[4] S. P. Singh, T. Jaakkola, and M. I. Jordan, "Learning without state-estimation in partially observable markovian decision processes," in *Proceedings of the 11th International Conference on Machine Learning*, 1994, pp. 284–292.

[5] J. Loch and S. Singh, "Using eligibility traces to find the best memoryless policy in partially observable markov decision processes," in *Proceedings of the 15th International Conference on Machine Learning*, 1998, pp. 141–150.

[6] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvari, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Machine Learning*, vol. 38, no. 3, pp. 287–308, 2000.

[7] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuska, "Fuzzy approximation for convergent model-based reinforcement learning," in *IEEE International Conference on Fuzzy Systems, FUZZ-IEEE*, 2007.

[8] E. Schuitema, M. Wisse, and P. Jonker, "The design of LEO: a 2D bipedal walking robot for online autonomous reinforcement learning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, October 2010.