Safe Reinforcement Learning for Continuous Spaces through Lyapunov-Constrained Behavior

Sigurd A. FJERDINGEN [a,1], and Erik KYRKJEBØ [a]
[a] *Dept. of Applied Cybernetics, SINTEF ICT, Norway*

Abstract. This paper presents a safe learning strategy for continuous state and action spaces by utilizing Lyapunov stability properties of the studied systems. The reinforcement learning algorithm Continous Actor-Critic Learning Automation (CACLA) is combined with the notion of control Lyapunov functions (CLF) to limit the learning and exploration behavior to operate inside the stability region of the system to ensure safe operation at all times. The paper extends previous results for discrete action sets to take advantage of the more general continuous actions sets, and show that the continuous method is able to find a comparable solution to the best discrete action choices while avoiding the need for good heuristic choices in the design process.

Keywords. Safe learning, reinforcement learning, control Lyapunov function

Introduction

Robots able to improve and learn new behavior over time through interaction with the real physical world has long been a goal for researchers in cybernetics, computer science and electronics. Such systems have been identified as a key enabling technology for more autonomous robot systems in all major application areas; industrial production-, process- and manufacturing operations, in tackling societal health- and home-care challenges, in personal robot assistants and in professional service robot systems.

One computational approach to learning from interaction is denoted reinforcement learning (RL) and addresses the issue of when to do what, i.e. how the current situation of a system should be mapped to an appropriate behavior or action in order to maximise a reward received from the environment [1]. A robot that interacts with a stochastic process (the environment) modelled as a Markov decision process (MDP) will be given an immediate reward when performing an action. The objective is to discover which actions yield the most accumulated reward over time, and this is done by experimenting with the effects of different actions.

A robotic agent that can learn new behavior may come to behave unexpectedly since exploration is an inherent part of discovering new and possibly better solutions. When a real robot is acting on and reacting to the physical world, special attention must be paid to the reliability and safety of behaviors to ensure that no harm may come to the robot or its environment. Most reinforcement learning algorithms, however, are not concerned with operational safety bounds during learning and exploration. Only recently have dif-

[1]Corresponding Author: Sigurd A. Fjerdingen, SINTEF ICT Applied Cybernetics, 7465 Trondheim, Norway; E-mail: sigurd.fjerdingen@sintef.no.

ferent aspects of safety in relation to robot learning been addressed by researchers [2,3]. Perkins and Barto [4,5] have integrated the notion of Lyapunov stability [6] from the control literature with a principal reinforcement learning algorithm for discrete action sets, SARSA(λ) (see e.g. Sutton and Barto [1]), to create a learning algorithm that operates within the stability bounds of a Lyapunov function.

This paper extends the ideas of Perkins and Barto using only discrete action sets to the more general case of using both continuous state and action spaces to address the need of working with actions of a continuous nature in many robot systems. A Lyapunov-based control design is combined with the Continuous Actor-Critic Learning Automaton (CACLA) reinforcement learning algorithm [7,8] specifically suited for continuous state and actions spaces. The resulting controller is applied to the benchmark learning task as presented by Perkins and Barto [4] of swing-up and balancing of an inverted pendulum to compare and analyse the results.

The paper is organised as follows. Section 1 gives a quick introduction to reinforcement learning with special attention to algorithms for continuous state and action spaces. Section 2 introduces Lyapunov functions and relates it to reinforcement learning. The problem definition and experimental studies conducted in the present work are presented in Section 3, while Section 4 presents the results of the experimental studies. A discussion and conclusions on the results are given in Section 5.

1. Markov Decision Processes and Reinforcement Learning

A Markov decision process (MDP), which is used to model an agent's interaction with the environment in reinforcement learning, can be defined as a tuple $(\mathcal{S}, \mathcal{A}, R, T)$, where $\mathcal{S}$ is the set of all states on which the MDP evolves, $\mathcal{A}$ the set of available actions ($\mathcal{A}$ is in general dependent on $\mathcal{S}$, i.e. $\mathcal{A}(s)$ where $s \in \mathcal{S}$ is the current state), R the reward function, and $T(s, a, s') \in [0, 1]$ the transition probability function. This denotes the probability of moving from the current state $s \in \mathcal{S}$ to the resulting state $s' \in \mathcal{S}$ by applying action $a \in \mathcal{A}(s)$. In reinforcement learning problems, the reward function and transition probability function are treated as unknowns and thus ordinary dynamic programming solution approaches do not apply. For more details on Markov decision processes and reinforcement learning, see e.g. the excellent introductory book by Sutton and Barto [1].

Value functions are central to reinforcement learning, and may be formally defined as

$$V^{\pi}(s) = \mathrm{E}\left\{ \sum_{i=0}^{\infty} \gamma_d^i r_{t+i+1} \mid \pi, s_t = s \right\}. \tag{1}$$

This function describes the future cumulative discounted reward an agent expects to receive from time t and towards infinity by following its current policy π from the current state $s_t = s$, where r_t is the immediately received reward from the environment at time t and $\gamma_d \in (0, 1]$ is a discount factor that controls the effect of rewards received in the future. Intuitively, the value function describes how good it is to be in a state. States with higher values will lead to larger cumulative rewards when following the policy π mapping from states to actions, i.e. $a = \pi(s)$.

Whereas the value function $V^{\pi}(s)$ describes the value of being in a particular state s following a particular policy π, a corresponding action-value function $Q^{\pi}(s, a)$ takes into account which actions yield the most future cumulative reward for each state, and is therefore well suited for control problems. An action-value function describes the expected return starting at time t and state $s_t = s$ taking action $a_t = a$ and thereafter following the policy π. Its formal definition is given as

$$Q^{\pi}(s,a) = \mathrm{E}\left\{\sum_{i=0}^{\infty} \gamma_d^i r_{t+i+1} \mid \pi, s_t = s, a_t = a\right\}. \tag{2}$$

This formulation has been the basis for many RL-algorithms, including SARSA which is an acronym for State-Action-Reward-State-Action.

SARSA is a well-known on-policy temporal difference-based reinforcement learning algorithm for control problems. The algorithm tries to estimate the Q-function iteratively a given policy π. That is, if a policy π, selecting actions a based on the current state s ($a = \pi(s)$), is followed, the algorithm will converge to describing the value of taking action a in any state s and thereafter follow policy π. Sutton and Barto [1] state the algorithm as

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma_d Q(s',a') - Q(s,a)\right), \tag{3}$$

where (s,a) is the current state-action pair, (s',a') the next, r the numerical reward received when transitioning from s to s' using action a, and $\alpha \in (0,1]$ is a learning rate parameter controlling the convergence speed of the algorithm. The type of policy normally used for this algorithm has some stochastic component, as for instance the ϵ-greedy policy which simply states that the estimated highest valued action (highest Q-value) should be chosen with probability $1-\epsilon$ else a random action should be chosen, i.e. a policy following the Bernoulli distribution. Exploration is here an implicit part of the policy, as random actions are chosen with probability ϵ for each state. SARSA converges to an optimal policy for discrete and finite states and actions under the assumptions that all state-action pairs are visited an infinite number of times and that the policy converges to a greedy policy, i.e. a policy that always selects the action a with the highest Q-value for a given state s (for instance by decaying ϵ in an ϵ-greedy policy for each epsiode conducted).

1.1. Continuous States and Actions

Many traditional reinforcement learning algorithms assume discrete state and action sets, such as the SARSA algorithm mentioned previously. Some control problems, such as the one studied in the current work of tuning a continuous parameter of a Lyapunov control function (see Section 2), are continuous in their nature. Fundamentals concerning MDPs still hold for continuous spaces, see e.g. van Hasselt [9]; chapter 7. RL algorithms for continuous state spaces have been extensively studied by researchers in the last two decades [1,10,11,12,13]. Using parameterized function approximators to represent the state space has emerged as the method of choice. Commonly used methods include non-linear approximators such as multilayer perceptrons [9], and linear approximators such as radial basis function networks (RBFNs) [14] and tile coding [1].

Two differing approaches of handling continuous actions are prominent in RL literature. A numerical optimization method may be used on the estimated Q-value (e.g. Newton-Raphson or wire-fitting [15]), or using a special subset of RL algorithms often denoted as actor-critic methods. In this type of RL algorithm, the estimation of the value function V (the critic) is separated from the estimation of the policy (the actor). The approach has been around for some time (see e.g. the 1977 article by Witten [16]), but has only recently gathered interest when problems using the direct determination approach became apparent [11].

1.1.1. Continuous Actor-Critic Learning Automaton

Van Hasselt and Wiering [8,9] present one such actor-critic method named CACLA. The value function is estimated by using the temporal difference errror

$$\delta = r + \gamma_d V(s') - V(s), \tag{4}$$

where γ_d is the discount factor, and thus becomes

$$V(s) = V(s) + \alpha_V \delta, \tag{5}$$

where $\alpha_V \in (0,1]$ is the learning rate of the value function. For continuous state spaces, the value function may be represented by a linear function approximator $\hat{V}_\theta(s)$ parameterized by a vector θ. Using gradient descent on the mean squared error between the experienced and currently estimated value function gives an update rule for the parameters as (see e.g. Sutton and Barto [1] for details)

$$\theta = \theta + \alpha_V \delta \nabla_\theta \hat{V}_\theta(s). \tag{6}$$

Note here that ∇_θ denotes the partial derivatives with respect to the parameter vector θ, i.e.

$$\nabla_\theta \hat{V}_\theta(s) = \left[\frac{\partial \hat{V}_\theta(s)}{\partial \theta_1}, \ldots, \frac{\partial \hat{V}_\theta(s)}{\partial \theta_N} \right]. \tag{7}$$

A stochastic policy $\Pr(a|s) = \pi(a|s;\phi)$ is often employed in actor-critic algorithms in order to create explorative behaviors, where ϕ denotes the parameter vector of a function approximator. In our case the policy is a stochastic policy employing the Gaussian distribution, given as

$$\pi(a|s;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(a-A^c_\phi(s))^2}{2\sigma^2}} \tag{8}$$

with mean $A^c_\phi(s) = \mu(s)$ approximated by a linear function approximator. In practice this means that it is the currently estimated mean of this Gaussian distribution that is stored by the function approximator. The parameter σ is used to control the amount of exploration for the policy. Note here that actor exploration may very well be directed by other means than Gaussian exploration [9].

The policy function approximator parameters for the CACLA actor, i.e. the parameters for $A^c_\phi(s)$, are updated by

$$\phi = \phi + \alpha_\pi \max(\operatorname{sgn}(\delta), 0) \left(a - A^c_\phi(s) \right) \nabla_\phi A^c_\phi(s), \tag{9}$$

where $\alpha_\pi \in (0,1]$ is the actor learning rate, and $\operatorname{sgn}(x)$ denotes the signum function returning $+1$ for $x > 0$, 0 for $x = 0$ and -1 for $x < 0$.

CACLA has been compared to the state of the art evolutionary method CMA-ES (Covariance Matrix Adaptation Evolution Strategy) [17] with favorable results [9], which again has compared favorably to other reinforcement learning methods such as NAC (Natural Actor Critic) [18]. This indicates that selecting the CACLA approach as the RL method of choice for continuous spaces when integrating with Lyapunov control functions should provide representative results.

The reader is referred to van Hasselt [8,9] for a more detailed description of the CACLA algorithm, and Fjerdingen et al [7] for practical usage and comparisons of CACLA to general actor-critic algorithms based on gradient descent.

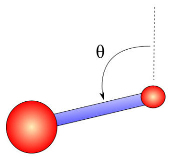


Figure 1. The pendulum environment. An actuater situated at the smallest of the two circles may swing the pendulum either left or right.

2. Safe Learning using Lyapunov Constraints

Lyapunov functions [6] are a fundamental mathematical tool for analyzing the stability of dynamical systems in control theory. The Lyapunov function L is a scalar energy-like function for a system - a generalization of the energy concept for mechanical systems - that can be determined to be asymptotically *stable* if the Lyapunov function decreases. This means that all solutions of the dynamical system starting out near an equilibrium point will converge to this point. For the pendulum problem as described in Section 3, the system has two equilibrium points; an asymptotically stable point downwards and an unstable equilibrium point upwards where any starting point near the upwards equilibrium point will make the move away from the point and increase the energy in the system.

The construction of a Control Lyapunov Function (CLF) can be used to test whether any dynamical system can be made asymptotically stable when subjected to a control input. A CLF is a positive-definite radially unbounded scalar Lyapunov-function L dependent on both *states* and *control inputs* that ensures that $\dot{L} < 0$. Thus, for any state we can find an action that reduces the "energy" in the system such that the Lyapunov-function has negative derivatives along the system trajectories. A safe and reliable reinforcement learning strategy should thus constrain all action choices to be within the appropriate control Lyapunov function.

3. Experiments

This section describes the simulation setup used to validate the previously described algorithm for safe reinforcement learning in continuous spaces. Simulations using the proposed algorithm are set up using the simulated inverted pendulum task described by Perkins and Barto [4] and compared to an implementation of the SARSA algorithm comparable to that of Perkins and Barto. This simulator is used in order to qualitatively compare results to previously obtained results.

3.1. Pendulum Problem Definition

The goal for the pendulum task is to swing up and balance a single-link pendulum under the effects of gravity, using a rotational actuator centered at the fulcrum of the pendulum, in the least possible time. The pendulum task is visualized in Figure 1, and is governed by the dynamics equation

$$\ddot{\theta} = \sin(\theta) + u, \tag{10}$$

where θ is the angular position of the pendulum using the vertical upright position as origin and u is the control torque. This equation assumes a pendulum mass of 1 and length 1, gravity of strength 1, and no damping. θ is assumed to stay in the range $[-\pi, \pi]$, since the states are identical for $(\theta + k2\pi)$ for $k \in \{0, \pm 1, \pm 2, \ldots\}$. The problem to optimize is to control the pendulum from hanging straight downwards to a near-upright, near-stationary position in minimum time. Perkins and Barto state this goal formally as $G_1 = \{(\theta, \dot{\theta}) \mid ||(\theta, \dot{\theta})||_2 \leq 0.01\}$. $\theta = 0$ is here defined as the upright position of the pendulum.

The implemented solver is based on the explicit Runge-Kutta (4,5) formula using the Dormand-Prince pair (the variable time-step MATLAB `ode45` solver) using fixed interrupts for each 0.25 seconds where control inputs may be changed, i.e. a controller frequency of 4 Hz. Note that these discrete controller intervals at the same time make the environment appear as a deterministic MDP. Control torque is bounded by $|u_{\max}| \leq 0.225$, which is too little torque to get the pendulum from a downwards hanging position to G_1 in one swing. The learner must therefore learn to swing the pendulum back and forth in order to gather enough momentum to swing it upright.

3.2. Lyapunov-Constrained Controller

The mechanical energy of the pendulum can be formulated as

$$\mathrm{M}(\theta, \dot{\theta}) = 1 + \cos(\theta) + \frac{1}{2}\dot{\theta}^2. \tag{11}$$

The total mechanical energy of the pendulum equals 2 when the pendulum is upright and stationary ($(\theta, \dot{\theta}) = 0$). The time derivative of the mechanical energy is then given by

$$\dot{\mathrm{M}}(\theta, \dot{\theta}) = -\sin(\theta)\dot{\theta} + \dot{\theta}\ddot{\theta} = \dot{\theta}u. \tag{12}$$

Perkins and Barto propose a controller based on this equation guaranteed to increase the mechanical energy of the pendulum at each time step. See Perkins [5] Theorem 6.1 and Appendix A for a detailed explanation and proof for this controller. The controller is given formally as [4]

$$u = \mathrm{MEA}(\theta, \dot{\theta}, w) = \begin{cases} \mathrm{sgn}(\dot{\theta})w & \text{if } |\dot{\theta}| > \dot{\epsilon} \\ & \vee\ (0 < \dot{\theta} < \dot{\epsilon}\ \wedge\ \theta \notin E_1) \\ & \vee\ (-\dot{\epsilon} < \dot{\theta} < 0\ \wedge\ \theta \notin E_2) \\ \frac{1}{2}\mathrm{sgn}(\dot{\theta})w & \text{if } (0 < \dot{\theta} < \dot{\epsilon}\ \wedge\ \theta \in E_1) \\ & \vee\ (-\dot{\epsilon} < \dot{\theta} < 0\ \wedge\ \theta \in E_2) \\ \mathrm{sgn}(\theta)w & \text{if } \dot{\theta} = 0, \end{cases} \tag{13}$$

where $\dot{\epsilon} \in \mathbb{R} > 0$ is set to 0.1 in the experiments. Perkins and Barto label this algorithm 'Modified Energy Ascent', and the intuitive difference between this controller and a normal energy ascent controller is that this controller switches from using w to $\frac{1}{2}w$ at the sets E_1 and E_2 surrounding equilibrium points where w equals the effects of gravity. These equilibrium points may result in stagnation of the pendulum, and Perkins and Barto avoid them by reducing the applied torque by 50%. The sets E_1 and E_2 are by Perkins and Barto given as

$$\begin{aligned} E_1 &= [-\pi + \phi - \epsilon,\ -\pi + \phi + \epsilon)\ \cup\ [-\phi - \epsilon,\ -\phi + \epsilon) \\ E_2 &= (\phi - \epsilon,\ \phi + \epsilon]\ \cup\ (\pi - \phi - \epsilon,\ \pi - \phi + \epsilon], \end{aligned} \tag{14}$$

where $\epsilon = \frac{1}{2}(\arcsin(w) - \arcsin(\frac{1}{2}w))$ and $\phi = \arcsin(w)$. Perkins and Barto define a Lyapunov function $L(\theta, \dot{\theta}) = 2 - \mathrm{M}(\theta, \dot{\theta})$ and an accompanying goal region $G_2 = \{(\theta, \dot{\theta}) \mid \mathrm{M}(\theta, \dot{\theta}) \geq 2\}$. The state of the pendulum is now guaranteed to descend on L outside of G_2, which makes it a Lyapunov function for the system. Applying zero torque after the pendulum reaches G_2 further ensures that the pendulum reaches the goal state G_1. In the experiments the pendulum torque has been set to $u = -100\ \mathrm{sgn}(\dot{\theta})(\mathrm{M}(\theta, \dot{\theta}) - 2)$

instead of 0 when the mechanical energy increased above 2 in order to dissipate possible extra energy due to numerical inaccuracies.

Using a SARSA algorithm similar to Perkins and Barto, the learner is allowed to select between two parameter values for w of the MEA control strategy as illustrated in their experiment 1 [4]. These two parameter values are either $w = u_{\max}$ or $w = \frac{1}{2}u_{\max}$.

In this paper we propose to integrate the CACLA algorithm within the Lyapunov learning framework. Referring to Section 1, the actor part of the algorithm is allowed to learn the optimal w parameter instead of switching between two discrete levels for the parameter. The actor is allowed to learn any $w \in \mathbb{R} > 0$. This means that values larger than $u_{\max}$ may be output from the MEA function. In our implementation, the output from the MEA-function is therefore stated as

$$u = \Gamma(\text{MEA}(\theta, \dot{\theta}, w)), \tag{15}$$

where $\Gamma(\cdot)$ is a function that caps the MEA at $u_{\max}$, i.e.

$$\Gamma(x) = \min(x, u_{\max}). \tag{16}$$

Allowing $w > u_{\max}$ has an effect on situations where the MEA-function switches from $\text{sgn}(\dot{\theta})w$ to $\frac{1}{2}\text{sgn}(\dot{\theta})w$ in that the $\frac{1}{2}$ fraction may be compensated for in some respects.

3.3. Specifics of the Implemented RL Algorithms

In the current simulations radial basis function networks (RBFNs) [19] are used as function approximators. Function approximation is a central part of any RL algorithm suited for continuous states and actions. RBFNs may be seen as a special kind of connectionist structure with three layers; an input layer, a hidden layer using a Gaussian activation function, and a linear output layer. The output of the network may be stated as

$$f(\mathbf{x}) = \sum_{i=1}^{N} \psi_i \rho(||\mathbf{x} - \mathbf{c}_i||), \tag{17}$$

where

$$\rho(||\mathbf{x} - \mathbf{c}_i||) = e^{-\frac{1}{\sigma}||\mathbf{x}-\mathbf{c}||^2} = \prod_j e^{-\frac{1}{\sigma_j}(x_j - c_{ij})^2} \tag{18}$$

is the Gaussian activation function with center vector $\mathbf{c}$, width σ, ψ a tunable weight parameter, and N the number of activation functions. An advantage of using an RBFN as opposed to a general multilayer perceptron is that the bilateral Gaussian function makes the approximations local in nature instead of the global possibilities inherent in a unilateral activation function such as the sigmoid.

For the CACLA algorithm two RBFNs have been used, each with an input space equal to the pendulum task state space $(\theta, \dot{\theta})$. Activation functions are spaced equally with $\Delta\theta = 0.2$ and $\Delta\dot{\theta} = 0.2$ in the regions $\theta \in [-\pi, \pi]$ and $\dot{\theta} \in [-3, 3]$. The learning rate parameter for the actor is set to $\alpha_\pi = 0.1$, and for the critic to $\alpha_V = 0.3$. The discount factor is set to $\gamma_d = 0.5$ and the Gaussian exploration factor $\sigma = 0.225$ at trial 0 with a linear decrease towards 0 at trial 1000. The total number of activation functions for each RBFN is 1054.

For the SARSA algorithm one RBFN is used to approximate the Q-function where an extra input dimension needs to be added relative to the actor-critic case, namely the action dimension. This dimension only contains activation functions along either the scalar 1 or 2, each number representing one of the two possible actions to take at any

Table 1. End-times for the last trial (trial 1000) on average for the simulations visualized in Figure 2-3.

Experiment	**End-time average in seconds**
SARSA 1	37.99 s
SARSA 2	66.78 s
SARSA 3	20.88 s
SARSA 4	21.41 s
CACLA	20.50 s

time. The learning rate parameter is set to $\alpha = 0.3$ and the discount factor equal to the one used for CACLA. An ϵ-greedy exploration is used with $\epsilon = 0.1$ for trial 1 with a linear decrease towards 0 for trial 1000. The total number of activation functions for the RBFN is 2108.

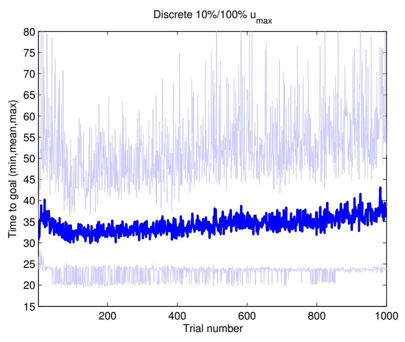
Reinforcement is sparse in the experiments, only given implicitly by giving $r = -1$ for each time-step. This means that the earlier a trial ends, the less total negative reward the learner will have received. A denser alternative could have been to give rewards based on the distance from the G_1 region, but we elect to keep our experiments in accordance with Perkins and Barto's experiments in this respect.

4. Results

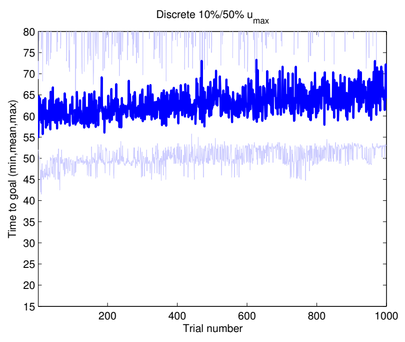
Figure 2 visualizes the result from four different experiments, each with different choices of the two possible actions to choose from (different values for w). SARSA simulation 3 (Figure 2(c) uses the same action values as used by Perkins and Barto in their work). Using the CACLA approach instead, the algorithm is allowed to choose any parameter value greater than zero as opposed to one of two discrete values. Figure 3 displays results from this experiment. Each experiment contains 1000 trials, and has been run 20 times to gather statistics. The figures show end-time results including exploration; only trial 1000 does not include any explorative behavior. Table 1 summarizes the final end-times for all five simulations. Figure 4 shows a comparison between one run of trial 1000 for the CACLA simulation and the SARSA simulation 3.

5. Conclusions

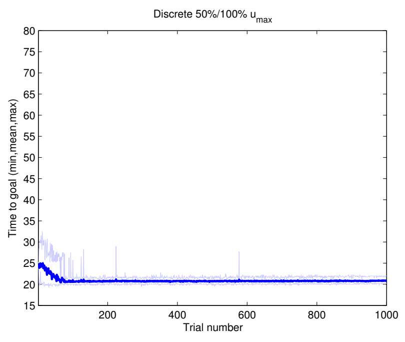
Results from Table 1 indicate that the optimal end-time of the discrete SARSA algorithm is heavily dependent on the choice of action values. Simulation 3 and 4 have apparently implemented better selection choices than simulation 1 and 2. This is also expected, as limiting the choice of actions to take should severely affect the possible optimal solution. The CACLA approach seems to find a comparable solution to the best action choices of the SARSA method, but without the need to make heuristic choices as to the action values to use. A common downside to RL algorithms aimed at continuous action spaces is that they are expected to converge at a considerably slower rate than a discrete counterpart employing a *correct* heuristic discretization. Figure 3 partly confirms the assumption of slower convergence (trial 1-400), but at the same time results from Table 1 oppose the assumption as the algorithm converges to a soltion comparable to the best SARSA solutions. Normally we would expect a significant amount of extra trials to converge to similar results since the problem of finding optimal solutions in continuous spaces is more complex than that of finding an optimal solution when only two choices are proposed. Looking at the parameter choices w for both algorithms, however, we see



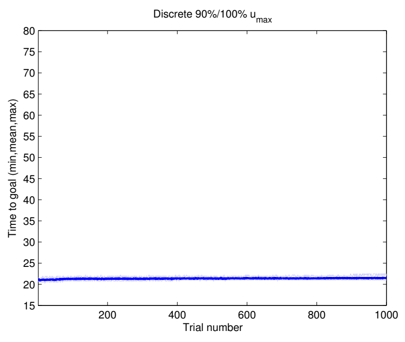
(a) SARSA experiment 1: $w = \{x \cdot u_{\max} \mid x = 0.1, 1.0\}$.



(b) SARSA experiment 2: $w = \{x \cdot u_{\max} \mid x = 0.1, 0.5\}$.



(c) SARSA experiment 3: $w = \{x \cdot u_{\max} \mid x = 0.5, 1.0\}$.



(d) SARSA experiment 4: $w = \{x \cdot u_{\max} \mid x = 0.9, 1.0\}$.

Figure 2. Simulations using the discrete SARSA algorithm using different selection possiblities for the w parameter, i.e. limiting the action set to different values. Experimentation is handled through ϵ-greedy exploration using a start parameter of 0.1, and linearly decreasing towards 0 indexed by the trial number. The figures show the minimum, mean, and maximum time to goal.

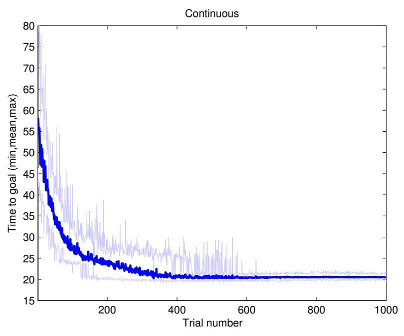
that the CACLA approach uses $w > u_{\max}$ a lot of the time, as shown in Figure 4. The MEA-algorithm proposed by Perkins and Barto contains a switch from full torque to half torque to avoid certain problematic areas, as previously discussed. The CACLA method is allowed to counteract the suboptimality of this heuristic switch in order to get a more optimal solution at a faster rate. This is not possible to discover using heursitics-based discretization of the action space limited by $w = x,\ x \in (0,\ u_{\max}]$.

This paper has presented a safe reinforcement learning method for continuous state and action spaces through the use of control Lyapunov functions. Perkins and Barto's results are extended to continuous action spaces by employing the state-of-the-art reinforcement learning algorithm CACLA. A potential issue of dealing with transients occuring due to changing of the controller parameters at run-time, which may or may not lead to unstability, has not been addressed and remains a task for further research.

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. The MIT Press, London, England, 1998.

[2] P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108, 2005.



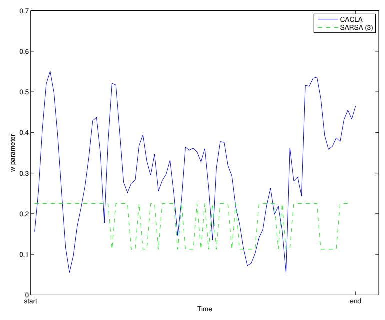


Figure 3. Simulations using the continuous CACLA algorithm. The w parameter is here only limited by $w = x,\ x > 0$. The pendulum torque is still limited by $u = y,\ y \in u_{\min}, u_{\max}$.

Figure 4. w parameter for CACLA (solid line) and SARSA simulation 3 (dotted line) during one run of the last trial.

[3] A. Hans, D. Schneegass, A. M. Schäfer, and S. Udluft. Safe exploration for reinforcement learning. In *ESANN'2008 proceedings, European Symposium on Artificial Neural Networks*, pages 143–148, 2008.

[4] T. J. Perkins and A. G. Barto. Lyapunov design for safe reinforcement learning. *Journal of machine learning research*, 3:803–832, 2002.

[5] T. J. Perkins. *Lyapunov Methods for Safe Intelligent Agent Design*. PhD thesis, University of Massachusetts, Amherst, 2002.

[6] H. K. Khalil. *Nonlinear Systems*. Prentice Hall, New Jersey, 3rd edition, 2002.

[7] S. A. Fjerdingen, E. Kyrkjebø, and A. A. Transeth. Auv pipeline following using reinforcement learning. In *Proceedings for the joint conference of ISR 2010 and ROBOTIK 2010*, pages 310–317. VDE Verlag GmbH, 2010.

[8] H. van Hasselt and M. A. Wiering. Reinforcement learning in continuous action spaces. In *Proceedings of the 2007 IEEE symposium on approximate dynamic programming and reinforcement learning*, pages 272–279, 2007.

[9] H. van Hasselt. *Insights in Reinforcement Learning*. PhD thesis, Dutch Research School for Information and Knowledge Systems, 2011.

[10] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 1995.

[11] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K. R. Muller, editors, *Advances in neural information processing systems*, volume 12 of *Advances in neural information processing systems*, pages 1057–1063. MIT Press, 2000.

[12] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*, 1993.

[13] J. N. Tsitsiklis and B. van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[14] X.-S. Wang, Y. H. Cheng, and W. Sun. Q learning based on self-organizing fuzzy radial basis function network. In J. Wang, Z. Yi, J. M. Zurada, B. L. Lu, and H. J. Yin, editors, *Advances in neural networks*, volume 3971 of *Lecture notes in computer science*, pages 607–615. Springer-verlag Berlin, 2006.

[15] C. Gaskett, D. Wettergreen, and A. Zelinsky. Reinforcement learning applied to the control of an autonomous underwater vehicle. In *Proceedings of the Australian conference on robotics and automation*, 1999.

[16] I. H. Witten. An adaptive optimal controller for discrete-time markov environments. *Information and Control*, 34(4):286–295, 1977.

[17] V. Heidrich-Meisner and Christian Igel. Similarities and differences between policy gradient methods and evolution strategies. In *Proceedings of the european symposium on artificial neural networks ESANN2008*, 2008.

[18] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Incremental natural actor-critic algorithms. In *Advances in Neural Information Processing Systems 20*, pages 105–112. MIT Press, Cambridge, MA, 2008.

[19] J. Park and J. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, 1991.