Delft University of Technology

Delft Center for Systems and Control

Technical report 08-031

Policy search with cross-entropy optimization of basis functions*

L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška

If you want to cite this report, please use the following reference instead:

L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Policy search with cross-entropy optimization of basis functions," *Proceedings of the 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2009)*, Nashville, Tennessee, pp. 153–160, Mar.–Apr. 2009.

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.51.19 (secretary)
fax: +31-15-278.66.79
URL: `http://www.dcsc.tudelft.nl`

*This report can also be downloaded via `http://pub.deschutter.info/abs/08_031.html`

Policy Search with Cross-Entropy Optimization of Basis Functions

Lucian Buşoniu, Damien Ernst, Bart De Schutter, and Robert Babuška

***Abstract*—This paper introduces a novel algorithm for approximate policy search in continuous-state, discrete-action Markov Decision Process (MDP). Previous policy search approaches have typically used ad-hoc parameterizations developed for specific MDPs. In contrast, the novel algorithm employs a flexible policy parameterization, suitable for solving general discrete-action MDPs. The algorithm looks for the best closed-loop policy that can be represented using a given number of basis functions, where a discrete action is assigned to each basis function. The locations and shapes of the basis functions are optimized, together with the action assignments. This allows a large class of policies to be represented. The optimization is carried out with the cross-entropy method and evaluates the policies by their empirical return from a representative set of initial states. We report simulation experiments in which the algorithm reliably obtains good policies with only a small number of basis functions, albeit at sizable computational costs.**

I. Introduction

Markov decision processes (MDPs) can be used to express important, general problems arising in various fields, including automatic control, operations research, economy, and computer science. Algorithms to solve general MDP s are therefore very promising for these fields. For instance, in automatic control, such an algorithm would provide a solution to the nonlinear, stochastic optimal control problem.

In an MDP , at each discrete time step, the controller measures the state of the process and applies an action according to a control policy.[1] As a result of this action, the process transits into a new state, possibly in a nonlinear and stochastic fashion, and a scalar reward signal is generated that indicates the quality of the transition. The controller measures the new state, and the whole cycle repeats. The control goal is to maximize the cumulative reward (the return) over the course of interaction [1], [2]. Because exact solutions can be found only for MDP s that have state and action spaces with a finite (and not too large) number of elements, approximate solutions have to be sought in general. In this paper, we only consider such approximate solutions.

The most widely used algorithms for solving MDP s rely on value functions, which give the returns from every state or state-action pair. The value function is used to compute a policy. Unfortunately, most state-of-the-art value-function techniques do not scale well to high-dimensional problems: they are typically applied to problems with only up to six continuous state variables [3]–[6]. Moreover, designing value function approximators is often a difficult, counterintuitive task [3]. Motivated by these shortcomings, direct policy search algorithms have been proposed [7]–[11]. These algorithms parameterize the policy and search for an optimal parameter vector which maximizes the return, without using a value function. In the literature, typically ad-hoc policy parameterizations with a few parameters are designed for specific problems, using intuition and prior knowledge about the optimal policy [8], [9]. On the other hand, in the area of value function approximation, significant efforts have been invested to develop techniques that automatically find good approximators, without relying on prior knowledge [3], [5], [6], [12]. Most of these techniques represent value functions using a linear combination of basis functions (BFs) , and automatically find BF s that lead to an accurate value function representation.

In this paper, we develop and evaluate a flexible policy approximator for direct policy search. Unlike previous approaches from the literature, this approximator can represent policies for a large class of MDP s. The algorithm works for continuous states and discrete (or discretized) actions. Inspired by the work on automatic BF construction for value function approximation, we propose to represent policies using N state-dependent BF s, where each BF is associated with a discrete action in a many-to-one fashion. The type of BF s and their number N are specified in advance and determine the approximation power of the representation. The locations and shapes of the BF s, together with the action assignments, are the parameters subject to optimization. The optimization criterion is a weighted sum of the returns from a set of representative initial states, where each return is computed with Monte Carlo simulations. The representative states together with the weight function can be used to focus the algorithm on important parts of the state space.

Mainly because of the rich policy parameterization, the performance may be a complicated function of the parameter vector, e.g., non-differentiable, non-convex, with many local optima. To address this problem, a gradient-free, global optimization technique is required; we select the cross-entropy (CE) method [13] as an illustrative example of such a technique. We employ CE optimization to optimize the BF s and the action assignments. We report numerical results, where CE policy search is used to control a double integrator and to balance a bicycle riding at a constant speed. Additionally, CE policy search is compared to a value-

Lucian Buşoniu, Bart De Schutter, and Robert Babuška are with the Center for Systems and Control of the Delft University of Technology, The Netherlands (email: i.l.busoniu@tudelft.nl, b@deschutter.info, r.babuska@tudelft.nl). Bart De Schutter is also with the Marine and Transport Technology Dept. of TU Delft. Damien Ernst is a Research Associate of the Belgian FNRS; he is affiliated with the Systems and Modeling Unit of the University of Liège, Belgium (email: dernst@ulg.ac.be).

This research is financially supported by the BSIK-ICIS project (grant no. BSIK03024), by the NWO Van Gogh grant VGP 79-99, and by the STW-VIDI project DWV.6188.

[1]Throughout the paper, control-theoretic terms and notations will be preferred to their artificial intelligence counterparts. For instance, the term 'controller' will be used instead of 'agent', and 'process' instead of 'environment'.

function technique in the double-integrator problem.

The remainder of this paper is structured as follows. Section II gives a brief overview of the literature related to our approach. Section III describes the MDP framework and the CE method for optimization. In Section IV, the proposed CE algorithm for policy search is introduced. Section V reports the results of our numerical experiments, and Section VI concludes the paper.

II. Related work

The use of the CE method for policy optimization was introduced in [10]. The authors of [11] proposed to find a policy with the model reference adaptive search, which is closely related to the CE method. Both works focus on solving finite, small MDP s, although they also propose solving large MDP s with parameterized policies. In contrast, we focus on flexible policy parameterizations to solve *continuous-state* MDP s. Additionally, we consider representative states associated with weights as a way to focus the optimization on important initial states, and to circumvent the need of estimating returns for every value of the state, which is impossible when the states are continuous. In [11], the return is optimized starting from a single initial state, whereas in [10] the returns from every (discrete) initial state are optimized.

The most widely-used technique for policy search is gradient-based optimization [7]–[9], [14]. Such work is based on the assumption that the locally optimal solution found by the gradient method is satisfactory, which may be true when the policy parameterization is simple and well suited for the particular problem. In contrast, for our rich policy parameterization, the optimization criterion is likely to have many unsatisfactory local optima and may also be non-differentiable. We select CE optimization as a representative technique from the plethora of the available gradient-free, global optimization methods. Another such method that has been applied to policy search is evolutionary computation [11].

Our policy parameterization is inspired by the techniques to automatically find BF s for value function approximation [3], [5], [6], [12]. However, rather than searching for value function BF s, we use optimization to search for *policy* BF s. In value function approximation, adapting the BF s while running the value function estimation algorithm can lead to loss of convergence, whereas in our policy search approach, the BF s can be adapted without causing convergence problems.

III. Preliminaries

A. Markov decision processes

In this section, MDP s are formally described and their optimal solution is characterized [1]. An MDP is defined by its state space X, its action space U, its transition probability function $f : X \times U \times X \to [0,\infty)$, and its reward function $\rho : X \times U \times X \to \mathbb{R}$. At each discrete time step k, given the state x_k, the controller takes an action u_k according to a control policy $h : X \to U$. The probability that the next state x_{k+1} belongs to a region $X_{k+1} \subset X$ of the state space is then $\int_{X_{k+1}} f(x_k,u_k,x')\mathrm{d}x'$. For any x and u, $f(x,u,\cdot)$ is assumed to define a valid density[2] of the argument '$\cdot$'. After the transition to x_{k+1}, a reward r_{k+1} is provided according to the reward function ρ: $r_{k+1} = \rho(x_k,u_k,x_{k+1})$. For deterministic MDP s, the transition probability function X is replaced by a simpler transition function, $\overline{f} : X \times U \to X$. In this case, the reward is completely determined by the current state and action: $r_{k+1} = \overline{\rho}(x_k,u_k)$, $\overline{\rho} : X \times U \to \mathbb{R}$.

The expected discounted return of an initial state x_0 under a policy h is:

$$R^h(x_0) = \lim_{K\to\infty} \mathrm{E}_{x_{k+1}\sim f(x_k,h(x_k),\cdot)} \left\{ \sum_{k=0}^{K} \gamma^k \rho(x_k,h(x_k),x_{k+1}) \right\} \quad (1)$$

where $\gamma \in [0,1)$ is the discount factor. The notation $a \sim p(\cdot)$ means that the random variable a is drawn from the density p. The goal is to find an optimal policy h^* that maximizes the expected return (1) for every initial state. For any MDP , there exists at least a deterministic optimal policy. Therefore, only *deterministic policies* will be considered in the sequel.

B. Cross-entropy optimization

This section briefly introduces the CE method for optimization [13]. Consider the following optimization problem:

$$\max_{a\in\mathscr{A}} s(a) \quad (2)$$

where $s : \mathscr{A} \to \mathbb{R}$ is the score function to maximize, and the variable a takes values in the domain $\mathscr{A}$. Let the maximum be denoted by s^*. The CE method for optimization maintains a density with support $\mathscr{A}$. In each iteration, a number of samples are drawn from this density and the score values for these samples are computed. A (smaller) number of samples that have the highest scores are kept, and the remaining samples are discarded. The density is then updated using the selected samples, such that at the next iteration the probability of drawing better samples is increased. The algorithm stops when the score of the worst selected sample no longer improves significantly.

Formally, a family of probability densities $\{p(\cdot;v)\}$ has to be chosen. This family has support $\mathscr{A}$ and is parameterized by v. In each iteration $\tau \geq 1$ of the CE algorithm, a number N_{CE} of samples is drawn from the density $p(\cdot;v_{\tau-1})$, their scores are computed, and the $(1-\rho_{\mathrm{CE}})$ quantile[3] λ_τ of the sample scores is determined, with $\rho_{\mathrm{CE}} \in (0,1)$. Then, a so-called associated stochastic problem is defined, which

[2]For simplicity, we will abuse the terminology by using the term 'density' to refer to probability density functions (which describe probabilities of continuous random variables), as well as to probability mass functions (which describe probabilities of discrete random variables).

[3]If the score values of the samples are ordered increasingly and indexed such that $s_1 \leq \cdots \leq s_{N_{\mathrm{CE}}}$, then the $(1-\rho_{\mathrm{CE}})$ quantile is: $\lambda_\tau = s_{\lceil (1-\rho_{\mathrm{CE}})N_{\mathrm{CE}} \rceil}$ where $\lceil\cdot\rceil$ rounds the argument to the next greater or equal integer number (ceiling).

involves estimating the probability that the score of a sample drawn from $p(\cdot;v_{\tau-1})$ is at least λ_τ:

$$\mathrm{P}_{a\sim p(\cdot;v_{\tau-1})}(s(a)\geq\lambda_\tau)=\mathrm{E}_{a\sim p(\cdot;v_{\tau-1})}\{I(s(a)\geq\lambda_\tau)\} \quad (3)$$

where I is the indicator function, equal to 1 whenever its argument is true, and 0 otherwise.

The probability (3) can be estimated by importance sampling. For the associated stochastic problem, an importance sampling density is one that increases the probability of the interesting event $s(a)\geq\lambda_\tau$. The best importance sampling density in the family $\{p(\cdot;v)\}$, in the sense of the smallest cross-entropy or Kullback-Leibler divergence, is given by the parameter that is the solution of:

$$\arg\max_v \mathrm{E}_{a\sim p(\cdot;v_{\tau-1})}\{I(s(a)\geq\lambda_\tau)\ln p(a;v)\} \quad (4)$$

An approximate solution of (4) is computed with the so-called stochastic counterpart:

$$v_\tau=\arg\max_v \frac{1}{N_{\mathrm{CE}}}\sum_{i_s=1}^{N_{\mathrm{CE}}} I(s(a_{i_s})\geq\lambda_\tau)\ln p(a_{i_s};v) \quad (5)$$

CE optimization then proceeds with the next iteration using the new density parameter v_τ (note that the probability (3) is never actually computed). The updated density aims at generating good samples with higher probability, thus bringing $\lambda_{\tau+1}$ closer to the optimum s^*. The goal is to eventually converge to a density that generates with very high probability samples close to optimal value(s) of a. The algorithm can be stopped when the improvement in the $(1-\rho_{\mathrm{CE}})$ quantile does not exceed $\varepsilon_{\mathrm{CE}}$ for d_{CE} successive iterations, or when a maximum number of iterations $\tau_{\max}$ is reached. The highest score among the samples generated in all the iterations is taken as the approximate solution of the optimization problem, and the corresponding sample as an approximate location of the optimum.

Under certain assumptions on $\mathscr{A}$ and $p(\cdot;v)$, the stochastic counterpart (5) can be solved analytically. One particularly important case when this happens is when $p(\cdot;v)$ belongs to the natural exponential family. For instance, when $\{p(\cdot;v)\}$ is the family of Gaussians parameterized by the mean η and the standard deviation σ (so, $v=[\eta,\sigma]^{\mathrm{T}}$), the solution v_τ of (5) consists of the mean and the standard deviation of the best samples, i.e., of the samples a_{i_s} for which $s(a_{i_s})\geq\lambda_\tau$.

While the convergence of CE optimization is not guaranteed in general, the algorithm is usually convergent in practice [13]. Convergence proofs are available for combinatorial optimization, where CE optimization provably converges with probability 1 to a unit mass density, which always generates samples equal to a single point [15]. Furthermore, this convergence point can be made equal to an optimal solution using an adaptive smoothing technique [15].

IV. Cross-entropy policy search

In this section, the proposed policy parameterization and optimization approach is described. Denote by D the number of state variables of the MDP (the dimension of X). In the sequel, it is assumed that the action space of the MDP is discrete and contains M distinct actions, $U_{\mathrm{d}}=\{u_1,\ldots,u_M\}$. The discrete set U_{d} can result from the discretization of an originally continuous action space U.

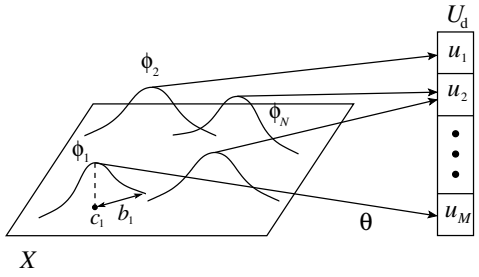


Fig. 1. A schematic representation of the policy parameterization. The vector θ associates the BFs to the discrete actions. In this example, the BFs are parameterized by their centers c_i and widths b_i, so that $\xi=[c_1^{\mathrm{T}},b_1^{\mathrm{T}},\ldots,c_N^{\mathrm{T}},b_N^{\mathrm{T}}]^{\mathrm{T}}$.

The policy parameterization is defined as follows. A set of N BF s over the state space is defined. The BF s are parameterized by a vector $\xi\in\Xi$, which typically gives their locations and shapes. Denote these BF s by $\phi_i(x;\xi):X\to\mathbb{R}$, $i=1,\ldots,N$, to highlight their dependence on ξ. The BF s are associated to the discrete actions by a many-to-one mapping. This mapping can be represented as a vector $\theta\in\{1,\ldots,M\}^N$ that associates each BF ϕ_i to a discrete action index $\theta(i)$, or equivalently to a discrete action $u_{\theta(i)}$. The expression $\theta(i)$ is used to denote the i-th element of θ. A schematic representation of this parameterization is given in Figure 1. The values in x of the BF s associated to each discrete action are summed up, and the action with the largest sum is chosen:

$$h(x)=u_{j^*},\ j^*=\arg\max_j \sum_{i\in\{i'\,|\,1\leq i'\leq N,\ \theta(i')=j\}}\phi_i(x;\xi)$$

The policy parameter vector $a=[\xi^{\mathrm{T}},\theta^{\mathrm{T}}]^{\mathrm{T}}$ ranges in the set $\mathscr{A}=\Xi\times\{1,\ldots,M\}^N$. CE optimization will be used to search for an optimal parameter vector that maximizes the score function:

$$s(\xi,\theta)=\sum_{x_0\in X_0} w(x_0)\widehat{R}^h(x_0) \quad (6)$$

where $\widehat{R}^h$ is the estimated return of the policy h given by ξ and θ, and X_0 is a given finite set of representative states, weighted by $w:X_0\to(0,1]$. The set X_0, together with the weight function w, will determine the performance of the resulting policy. Some problems only require to optimally control the system from a restricted set of initial states; X_0 should then be equal to this set, or included in it when the set is too large. Also, initial states that are deemed more important can be assigned larger weights. When all initial states are equally important, the elements of X_0 should be uniformly spread over the state space, and identical weights equal to $\frac{1}{|X_0|}$ should be assigned to every element of X_0 ($|\cdot|$ denotes set cardinality).

The return for each state in X_0 is estimated by Monte Carlo

simulations:

$$\widehat{R}^h(x_0) = \frac{1}{N_{\mathrm{MC}}} \sum_{i_0=1}^{N_{\mathrm{MC}}} \sum_{k=0}^{K} \gamma^k \rho(x_{i_0,k}, h(x_{i_0,k}), x_{i_0,k+1})$$

where $x_{i_0,0} = x_0$, $x_{i_0,k+1} \sim f(x_{i_0,k}, h(x_{i_0,k}), \cdot)$, and N_{MC} is the number of Monte Carlo simulations to carry out. Each Monte Carlo simulation i_0 uses a system trajectory that consists of K steps and is generated using the policy h. The system trajectories are generated independently, so the score computation is unbiased. To guarantee that an error of at most $\varepsilon_{\mathrm{MC}}$ is introduced by truncating the discounted sum of rewards after K steps, the value of K is chosen with:

$$K = \left\lceil \log_\gamma [\varepsilon_{\mathrm{MC}}(1-\gamma)/\left\|\rho\right\|_\infty] \right\rceil \tag{7}$$

where $\left\|\rho\right\|_\infty = \max_{x,u,x'} |\rho(x,u,x')|$ is assumed finite [10].

In order to define the associated stochastic problem (3) for the optimization problem of maximizing (6), it is necessary to choose a family of densities with support $\Xi \times \{1,\ldots,M\}^N$. The set Ξ is typically not discrete, so it is convenient to use separate densities for the two parts ξ and θ of the parameter vector. Denote the density for ξ by $p_\xi(\cdot; v_\xi)$, parameterized by v_ξ and with support Ξ, and the density for θ by $p_\theta(\cdot; v_\theta)$, parameterized by v_θ and with support $\{1,\ldots,M\}^N$. Let N_{v_ξ} be the number of elements in the vector v_ξ, and N_{v_θ} the number of elements in v_θ.

The CE algorithm for direct policy search is given as Algorithm 1. For easy reference, Table I collects the meaning of the parameters and variables of CE policy search. The stochastic counterparts in lines 9 and 10 of Algorithm 1 have been simplified, using the fact that the samples are already sorted in the ascending order of their scores. When $\varepsilon_{\mathrm{CE}} = 0$, the algorithm terminates when λ remains constant for d_{CE} consecutive iterations. When $\varepsilon_{\mathrm{CE}} > 0$, the algorithm terminates when λ improves for d_{CE} consecutive iterations, and these improvements do not exceed $\varepsilon_{\mathrm{CE}}$.

Algorithm 1 Cross-entropy policy search

Input:
dynamics f, reward function ρ, discount factor γ
representative states X_0 and their weight function w
density families $\{p_\xi(\cdot; v_\xi)\}, \{p_\theta(\cdot; v_\theta)\}$
initial density parameters $v_{\xi,0}$, $v_{\theta,0}$
parameters N, ρ_{CE}, c_{CE}, d_{CE}, $\varepsilon_{\mathrm{CE}}$, $\varepsilon_{\mathrm{MC}}$, N_{MC}, $\tau_{\max}$

1: set number of samples $N_{\mathrm{CE}} \leftarrow c_{\mathrm{CE}}(N_{v_\xi} + N_{v_\theta})$
2: $\tau \leftarrow 1$
3: **repeat**
4: Generate samples $\xi_1,\ldots,\xi_{N_{\mathrm{CE}}}$ from $p_\xi(\cdot; v_{\xi,\tau-1})$
5: Generate samples $\theta_1,\ldots,\theta_{N_{\mathrm{CE}}}$ from $p_\theta(\cdot; v_{\theta,\tau-1})$
6: Compute $s(\xi_{i_s}, \theta_{i_s})$ by (6), $i_s = 1,\ldots,N_{\mathrm{CE}}$
7: Reorder and reindex s.t. $s_1 \le \cdots \le s_{N_{\mathrm{CE}}}$
8: $\lambda_\tau \leftarrow s_{\lceil (1-\rho_{\mathrm{CE}})N_{\mathrm{CE}} \rceil}$
9: $v_{\xi,\tau} \leftarrow \arg\max_{v_\xi} \sum_{i_s=\lceil (1-\rho_{\mathrm{CE}})N_{\mathrm{CE}} \rceil}^{N_{\mathrm{CE}}} \ln p_\xi(\xi_{i_s}; v_\xi)$
10: $v_{\theta,\tau} \leftarrow \arg\max_{v_\theta} \sum_{i_s=\lceil (1-\rho_{\mathrm{CE}})N_{\mathrm{CE}} \rceil}^{N_{\mathrm{CE}}} \ln p_\theta(\theta_{i_s}; v_\theta)$
11: $\tau \leftarrow \tau + 1$
12: **until** $(\tau > d_{\mathrm{CE}}$ **and** $0 \le \lambda_{\tau-\tau'} - \lambda_{\tau-\tau'-1} \le \varepsilon_{\mathrm{CE}}$, for $\tau' = 1,\ldots,d_{\mathrm{CE}}-1)$ **or** $\tau > \tau_{\max}$

Output: $\widehat{\xi}^*, \widehat{\theta}^*$, the best sample; and $\widehat{s}^* = s(\widehat{\xi}^*, \widehat{\theta}^*)$

TABLE I
PARAMETERS AND VARIABLES FOR CE POLICY SEARCH.

Symbol	Meaning
N; M	number of BF s; number of discrete actions
ξ; θ	BF parameters; assignment of discrete actions to BF s
v_ξ; v_θ	parameters of the density for ξ; and for θ
N_{CE}	number of samples used at every CE iteration
ρ_{CE}	proportion of samples used in the CE updates
λ	$(1-\rho_{\mathrm{CE}})$ quantile of the sample performance
c_{CE}	multiple of the number of density parameters
d_{CE}	how many iterations λ should remain roughly constant
$\varepsilon_{\mathrm{CE}}$; $\varepsilon_{\mathrm{MC}}$	convergence threshold; precision in computing returns
N_{MC}	number of Monte Carlo simulations for each state
τ; $\tau_{\max}$	iteration index; maximum number of iterations

Many times it is convenient to use distributions with unbounded support (e.g., Gaussians) when the BF parameters are continuous. However, usually the set Ξ must be bounded, e.g., when ξ contains centers of radial BF s, which must remain inside a bounded state space. Whenever this situation arises, samples can be generated from the density with larger support, and those samples that do not belong to Ξ can be rejected and replaced by new samples. The procedure continues until N_{CE} valid samples are generated, and the rest of the algorithm remains unchanged. The situation is entirely similar for the discrete action assignments θ, when it is convenient to use a family of densities $p_\theta(\cdot; v_\theta)$ with a support larger than $\{1,\ldots,M\}^N$. An equivalent algorithm that uses all the samples can always be given, and therefore the theoretical basis of the CE optimization procedure is not affected by sample rejection.

The most important parameters in CE policy search for optimization are the number of samples N_{CE} and the proportion of best samples used to update the density, ρ_{CE}. The parameter c_{CE} is taken greater than or equal to 2, so that the number of samples is a multiple of the number of density parameters. The parameter ρ_{CE} can be taken around 0.01 for large numbers of samples, or larger, around $\ln(N_{\mathrm{CE}})/N_{\mathrm{CE}}$, if there are only a few samples ($N_{\mathrm{CE}} < 100$) [13]. The parameter $\varepsilon_{\mathrm{MC}} > 0$ can be chosen a few orders of magnitude smaller than the typical return obtained from the states in X_0. Since it does not make sense to impose a convergence threshold smaller than the precision of the score function, $\varepsilon_{\mathrm{CE}}$ should be chosen larger than or equal to $\varepsilon_{\mathrm{MC}}$. A good value is $\varepsilon_{\mathrm{CE}} = \varepsilon_{\mathrm{MC}}$. The number N of BF s determines the accuracy of the policy approximator, and a good value for N will depend on the problem. In Section V, we study the effect of varying N in an example problem. For deterministic MDP s, a single trajectory is simulated for every initial state in X_0, so $N_{\mathrm{MC}} = 1$. For stochastic MDP s, several trajectories should be simulated, $N_{\mathrm{MC}} > 1$, with a good value of N_{MC} depending on the MDP considered.

Next, we describe an instantiation of the CE policy search algorithm, using state-dependent Gaussian radial basis functions (RBFs) and a binary representation of the action assignments. This instantiation will be used in the examples

of Section V.

The (axis-parallel) Gaussian RBF s are given by:

$$\phi_i(x;\xi) = \exp\left[-\sum_{d=1}^{D}\frac{(x_d - c_{i,d})^2}{b_{i,d}^2}\right]$$

where D is the number of state variables, and each RBF ϕ_i is parameterized by its center $c_i \in X$ and its radius $b_i \in (0,\infty)^D$. The parameter vector ξ of the RBF s contains the N centers and radii: $\xi = [c_1^{\mathrm{T}},\ldots,c_N^{\mathrm{T}},b_1^{\mathrm{T}},\ldots,b_N^{\mathrm{T}}]^{\mathrm{T}}$, and is restricted to the domain $\Xi = X^N \times (0,\infty)^{DN}$. The total number of parameters is $2DN$.

To define the associate stochastic problem (3) for CE optimization, a scalar Gaussian density is used for each element of the vector ξ. Each such individual density is parameterized by its mean and standard deviation, so the complete density parameter v_ξ has $4DN$ elements. At every iteration, samples that do not belong to Ξ are rejected and replaced by new samples. As explained in Section III-B, the updated Gaussian density parameters $v_{\xi,\tau}$ in line (9) of Algorithm 1 can be computed analytically, as the mean and the standard deviation of the best samples $\xi_{\lceil(1-\rho_{\mathrm{CE}})N_{\mathrm{CE}}\rceil},\ldots,\xi_{N_{\mathrm{CE}}}$.

The assignments θ of discrete actions to BF s are represented in binary code. Each element $\theta(i)$ in the indices vector is represented using $N^{\mathrm{bin}} = \lceil \log_2 M \rceil$ bits, so that the binary representation of θ has NN^{bin} bits. Every bit is drawn from a Bernoulli distribution parameterized by its mean $\eta^{\mathrm{bin}} \in [0,1]$ (η^{bin} gives the probability of selecting 1; the probability of selecting 0 is $1-\eta^{\mathrm{bin}}$). Whenever M is not a power of 2, bit combinations corresponding to invalid indices are rejected and generated again. Such a concatenation of Bernoulli distributions can converge a degenerate distribution that always generates samples equal to a precise optimum location. Because every bit has its own Bernoulli parameter, the total number of Bernoulli parameters v_θ is NN^{bin}. The Bernoulli distribution belongs to the natural exponential family, so the updated density parameters $v_{\theta,\tau}$ in line 10 of Algorithm 1 can be computed analytically, as the mean of the best samples in their binary representation.

Next, we briefly examine the complexity of the above instantiation of CE policy search. The total number of samples used is $N_{\mathrm{CE}} = c_{\mathrm{CE}}(N_{v_\xi} + N_{v_\theta}) = c_{\mathrm{CE}}(4DN + NN^{\mathrm{bin}})$. The largest amount of computation is spent by the algorithm in the Monte Carlo simulations required to estimate the score of each sample. Neglecting therefore the other computations, the complexity of one iteration of the algorithm is at most:

$$t_{\mathrm{step}}[c_{\mathrm{CE}}N(4D + N^{\mathrm{bin}})\,|X_0|\,N_{\mathrm{MC}}K] \tag{8}$$

where K is the maximum length (7) of each trajectory, and t_{step} is the time needed to compute $h(x)$ for a fixed x and to simulate the controlled system for one time step.

V. Experimental studies

In the sequel, numerical experiments are carried out to assess the performance of CE policy search. Two examples are considered: a double-integrator problem, the simplicity of which allows extensive experiments to be run, and a more realistic bicycle balancing problem.

A. Discrete-time double integrator

In this section, a simple example is used to evaluate the proposed CE policy search, and to compare it with an algorithm relying on value functions. The example involves the optimal control of a double integrator, and is chosen so that optimal and near-optimal trajectories from any initial state terminate in a small number of steps. This property permits a significant reduction of the cost (8), and therefore extensive simulation experiments can be run.

The double-integrator problem is deterministic with a two-dimensional continuous state space $X = [-1,1]\times[-0.5,0.5]$, a binary action space $U_{\mathrm{d}} = \{-0.1, 0.1\}$, and the following dynamics:

$$x_{k+1} = f(x_k,u_k) = \mathrm{sat}\left\{[x_{1,k} + x_{2,k}, x_{2,k} + u_k]^{\mathrm{T}}, -x_{\max}, x_{\max}\right\}$$

where $x_{\max} = [1, 0.5]^{\mathrm{T}}$ and 'sat' denotes saturation, which is used to restrict the evolution of the state to X. The states for which $|x_1| = 1$ are terminal. Applying any action in a terminal state brings the process back to the same state, with a zero reward. The goal is to drive the position x_1 to either boundary of the interval $[-1,1]$ (i.e., to a terminal state), in such a way that at the moment when x_1 reaches the boundary, the speed x_2 as small as possible in magnitude. This goal is expressed by the reward function:

$$r_{k+1} = \rho(x_k,u_k) = -(1 - |x_{1,k+1}|)^2 - x_{2,k+1}^2 x_{1,k+1}^2$$

The discount factor γ is set to 0.95.

To apply CE policy search, the following grid of representative initial states was chosen to compute the score (6):

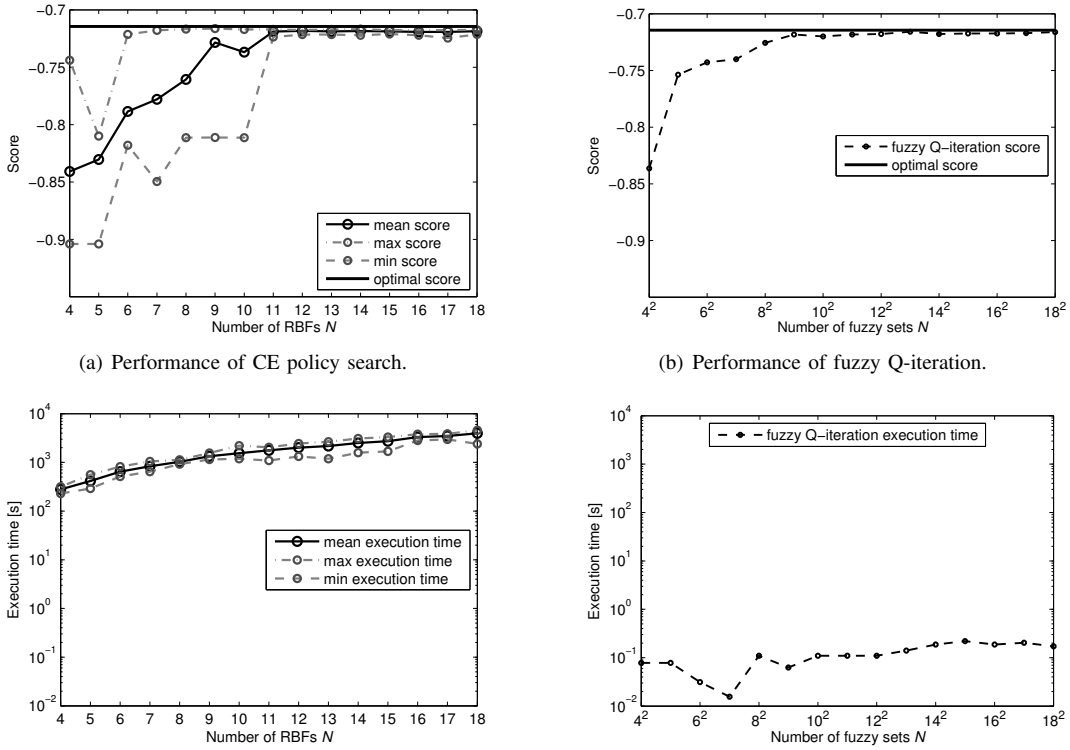
$$X_0 = \{-1,-0.9,\ldots,1\}\times\{-0.5,-0.3,-0.1,0,0.1,0.3,0.5\}$$

The states were equally weighted using $w(x_0) = 1/|X_0|$ for any x_0. The parameter settings for the algorithm were $c_{\mathrm{CE}} = 5$, $\rho_{\mathrm{CE}} = 0.05$, $\varepsilon_{\mathrm{CE}} = \varepsilon_{\mathrm{MC}} = 0.005$, $d_{\mathrm{CE}} = 5$, $\tau_{\max} = 50$. Little or no tuning was necessary to choose these values. Because the system is deterministic, $N_{\mathrm{MC}} = 1$. For all the experiments presented below, the maximum number of 50 iterations was never reached before convergence.

With these parameter settings, CE policy search was run while gradually increasing the number N of BF s from 4 to 18. Ten independent runs were performed for every N. The mean, maximum, and minimum performance obtained across these ten runs are given in Figure 2(a). For comparison, the exact optimal score for X_0 was computed using a brute-force search for optimal action sequences, which required 12841 s of CPU time to complete.[4]

Moreover, CE policy search is compared with the model-based, fuzzy Q-iteration algorithm [16]. Fuzzy Q-iteration is representative for the class of algorithms relying on value functions. Fuzzy Q-iteration computes an approximately optimal state-action value function (Q-function) which gives

[4]All the computation times reported in this paper were recorded while running the algorithms in MATLAB 7.1 on a PC with an Intel Pentium IV 3 GHz CPU, 512 MiB RAM, and using Windows XP.



(a) Performance of CE policy search.

(b) Performance of fuzzy Q-iteration.

(c) Execution time of CE policy search.

(d) Execution time of fuzzy Q-iteration.

Fig. 2. Comparison between CE policy search and fuzzy Q-iteration for the double integrator. Note the difference in the scale of the horizontal axes between (a, c) (linear) and (b, d) (quadratic).

approximately optimal returns from every state-action pair. This Q-function is then used to compute an approximately optimal policy. To approximate the Q-function, fuzzy Q-iteration uses discretized actions and a fuzzy partition of the state space (which can be regarded as a collection of BF s). We select equidistant, triangular BF s for every state variable. The two-dimensional BF s are then computed as the products of each combination of one-dimensional BF s [16]. This leads to bilinear interpolation over the state space, on the equidistant grid of triangle vertices. The evolution of the fuzzy Q-iteration performance with the number of BF s is shown in Figure 2(b). The number of BF s for each state variable was gradually increased from 4 to 18. The total number of two-dimensional BF s is the *square* of this number.

In Figure 2(a), CE policy search gives a large variance in the performance when $N = 4, \ldots, 10$, although the mean performance is increasing and a near-optimal performance is reached in some experiments starting from $N = 7$. Starting from $N = 10$, the algorithm consistently and reliably obtains a near-optimal performance. In contrast, in Figure 2(b) fuzzy Q-iteration obtains a near-optimal performance only starting from 9 BF s *on each axis*, which corresponds to a (much larger) total number of 81 BF s. So, to obtain a similar performance to CE policy search with N BF s, fuzzy Q-iteration requires a number of BF s roughly equal to the square of this number. This difference is mainly due to the fact that the BF s used by fuzzy Q-iteration are equidistant and identically shaped, whereas the CE algorithm optimizes the shapes and locations of the BF s.

Figures 2(c) and 2(d) compare the execution time of CE policy search and fuzzy Q-iteration, as a function of N. The execution time for fuzzy Q-iteration is much smaller than for CE policy search. This is because an iteration of CE policy search is much more computationally expensive than a fuzzy Q-iteration. So, in this problem, *optimizing* the BF s for the *policy representation* leads to a better performance than using *equidistant, identical* BF s for *value function approximation*, but at significantly higher computational costs. This indicates that CE policy search should preferably be used when a flexible policy approximator having a fixed complexity (determined by the number N of BF s) has to be found, and the computational costs to optimize this fixed-complexity approximator are not a concern.

B. Bicycle balancing

In this section, the CE policy search algorithm is applied to the more involved, stochastic bicycle balancing problem [4], [5], [17]. In this problem, a bicycle that rides at constant speed on a horizontal surface has to be prevented from falling, and the control actions are affected by noise. The steering column of the bicycle is vertical, which implies that the bicycle is not self-stabilizing. Instead, it has to be actively stabilized to prevent it from falling. The state vector is $[\omega, \dot{\omega}, \alpha, \dot{\alpha}]^{\mathrm{T}}$, where ω [rad] is the roll angle of the bicycle measured from the vertical axis, α [rad] is the

angle of the handlebar, taken equal to 0 when the handlebar is in its neutral position, and $\dot{\omega}$, $\dot{\alpha}$ [rad/s] are the respective angular velocities. The control variables are the displacement $\delta \in [-0.02, 0.02]$ m of the bicycle-rider common center of mass perpendicular to the plane of the bicycle, and the torque $T \in [-2, 2]$ Nm applied to the handlebar. The displacement δ is affected by additive noise drawn from a uniform distribution over the interval $[-0.02, 0.02]$ m. We refer the reader to [5] for more details about the bicycle problem, including its dynamical model.

The bicycle is considered to have fallen when the roll angle is larger than $\frac{12\pi}{180}$ in either direction, in which case a terminal, failure state is reached, and a reward of -1 is generated. All other rewards are 0. The discount factor is $\gamma = 0.98$. The actions are discretized as follows: the rider displacement is discretized into $\{-0.02, 0, 0.02\}$, and the torque on the handlebar into $\{-2, 0, 2\}$. This provides a discrete action space that is sufficient to balance the bicycle.

In order to study the influence of the set of representative states X_0 on the performance of the resulting policies, experiments were run for two different sets of representative states. In both cases, the initial states were uniformly weighted (i.e., $w(x_0) = 1/|X_0|$ for any $x_0 \in X_0$). Because we are mainly interested in the behavior of the bicycle starting from different initial rolls and roll velocities, the initial steering angle α and velocity $\dot{\alpha}$ are always taken equal to zero; this also prevents an excessive computational cost of the CE policy search. The first set of representative states contains a few evenly-spaced values for the initial roll of the bicycle, and the rest of the state variables are initially zero:

$$X_{0,1} = \left\{ \tfrac{-10\pi}{180}, \tfrac{-5\pi}{180}, \ldots, \tfrac{10\pi}{180} \right\} \times \{0\} \times \{0\} \times \{0\}$$

The second set is the cross-product of a finer roll grid and a few values of the roll velocity:

$$X_{0,2} = \left\{ \tfrac{-10\pi}{180}, \tfrac{-8\pi}{180}, \ldots, \tfrac{10\pi}{180} \right\} \times \left\{ \tfrac{-30\pi}{180}, \tfrac{-15\pi}{180}, \ldots, \tfrac{30\pi}{180} \right\} \times \{0\} \times \{0\}$$

For $X_{0,1}$, the optimal score is 0, because a good policy can always prevent the bicycle from falling for any initial roll value in $X_{0,1}$. This is no longer true for $X_{0,2}$: when ω and $\dot{\omega}$ have the same sign and are too large in magnitude, the bicycle cannot be prevented from falling by any control policy. Also, the initial roll velocities are not taken too large in magnitude to prevent including in $X_{0,2}$ too many states from which falling is unavoidable.

A number of $N = 8$ RBF s was selected,[5] and $N_{\text{MC}} = 10$ trajectories were simulated from every initial state to compute the Monte Carlo score (N_{MC} was not selected too large to keep the computational requirements of the algorithm manageable). The rest of the parameters were the same as in the double-integrator example, i.e., $c_{\text{CE}} = 5$, $\rho_{\text{CE}} = 0.05$, $\varepsilon_{\text{CE}} = \varepsilon_{\text{MC}} = 0.005$, $d_{\text{CE}} = 5$, $\tau_{\max} = 50$. The maximum number of 50 iterations was never reached before convergence. Ten independent experiments were run for each of the two sets of representative states.

TABLE II
RESULTS OF CE POLICY SEARCH FOR BICYCLE BALANCING.

Results	Performance		Execution time [s]	
	$X_{0,1}$	$X_{0,2}$	$X_{0,1}$	$X_{0,2}$
maximum	0	-0.2096	27519	218589
mean	0	-0.2108	22697	180856
minimum	0	-0.2123	16063	153201

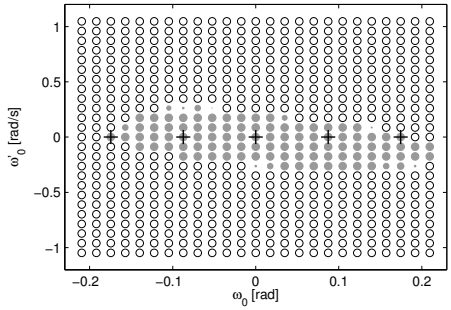
The performance of the resulting policies, as well as the execution time of CE policy search, are reported in Table II. It can be seen that all the scores for $X_{0,1}$ are optimal. It is not as easy to interpret the scores for $X_{0,2}$; see Figure 3 and the explanation below for a more intuitive representation of the obtained performance. The execution times are predictably much larger than for the simple, deterministic double integrator. Also, the execution times are one order of magnitude larger for $X_{0,2}$ than for $X_{0,1}$; this difference is due to the similar difference between the numbers of representative states in the two sets: $|X_{0,2}| = 55$ and $|X_{0,1}| = 5$.

Figure 3 illustrates the quality of some typical policies obtained by CE policy search, by verifying how they generalize to initial states that do not belong to X_0 (i.e., how they generalize). The initial steering angle and velocity are always 0. A number of 10 controlled trajectories are simulated for every initial state. The length of each trajectory is 50 s. This length was chosen to verify whether the bicycle is balanced robustly for a long time. It is roughly 10 times longer than the length of the trajectory used to evaluate the score during the optimization procedure, which was 4.56 s (corresponding from $K = 456$, which was computed by (7) with $\varepsilon_{\text{MC}} = 0.005$). The larger set $X_{0,2}$ of initial states is beneficial, because it leads to the bicycle being balanced for a much larger portion of the $(\omega, \dot{\omega})$ plane. Figure 3(b) also suggests that some states in $X_{0,2}$ are failure states, from where the bicycle falls regardless of the policy. These states correspond to values of ω and $\dot{\omega}$ that are large in magnitude and have the same sign. This explains why all the score values obtained for $X_{0,2}$ are negative in Table II.

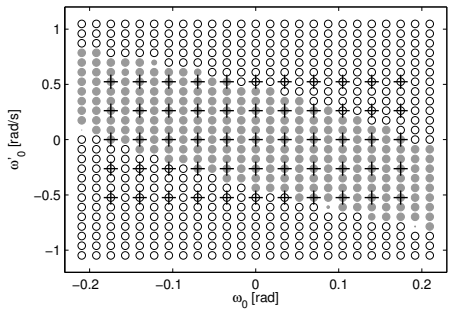
Other approaches to control the bicycle found in the literature use a more involved version of the problem, where the bicycle also has to ride to a target position in addition to being kept upright [4], [5]. Therefore, our results cannot be directly compared with the results of those approaches. Nevertheless, in [4], a total number of 100 *hand-tuned* BF s are used to approximate Q-functions in the context of policy iteration, whereas in our experiments as few as 8 *automatically optimized* BF s suffice to obtain a reliable performance.[6] In [5], a decision-tree approximator is automatically constructed in order to approximate the Q-function in the context of value iteration. The derivation automatically produces a large number of BF s, in the order of tens of thousands or more.

[5] Experiments on a deterministic version of the bicycle (not reported here) indicated that $N = 8$ is sufficient to adequately represent good policies.

[6] We also applied the algorithm of [4] to bicycle balancing using equidistant BF s, and failed to obtain good results with much larger numbers of BF s (more than 5000). These results are not reported here.



(a) Generalization for $X_{0,1}$.



(b) Generalization for $X_{0,2}$.

Fig. 3. Generalization of two typical policies over initial states not in X_0. White markers mean the bicycle was never balanced starting from that initial state; gray markers are proportional in size with the number of times the bicycle was properly balanced out of 10 experiments. Black crosses mark the initial states in X_0.

VI. Conclusions and Future Work

In this paper, we have introduced and evaluated a CE policy search technique for continuous-state, discrete-action MDP s. This novel technique uses a flexible policy parameterization, inspired by the work on automatic construction of BF s for value function approximation. The technique optimizes a weighted sum of the returns from a set of representative initial states, where each return is computed with Monte Carlo simulations. CE policy search has performed well both in a double-integrator example and in the more difficult problem of balancing an unstable bicycle. The algorithm has reliably offered a good performance, using only a small number of BF s to represent the policy. However, this performance has come at a large computational cost, e.g., for the double integrator, several orders of magnitude higher than the value-function based fuzzy Q-iteration. This indicates that CE policy search should preferably be used in situations where a flexible policy approximator having a fixed complexity (determined by the number N of BF s) has to be found, and the computational costs to optimize this fixed-complexity approximator are not a concern.

The theoretical study of CE policy search is an important opportunity for further research. Convergence results for the CE method are unfortunately only available for combinatorial optimization [13], [15], whereas CE policy search also involves the optimization of continuous variables. The convergence results for the related model-reference adaptive search [11] require the restrictive assumption that the optimal policy parameter is unique. It is an open question how the computational costs of CE policy search compare with the costs of value-function based algorithms for higher-dimensional problems than the double integrator; further experimentation could help answering this question. Another interesting direction is extending CE policy search to work for continuous-action policy parameterizations.

While in this paper the CE method for optimization has been employed, there is in principle no obstacle to applying any optimization technique to determine good policy parameters. In particular, other meta-heuristic optimization techniques like genetic algorithms, tabu search, pattern search, etc., could be used.

References

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2007, vol. 2.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[3] R. Munos and A. Moore, "Variable-resolution discretization in optimal control," *Machine Learning*, vol. 49, no. 2-3, pp. 291–323, 2002.

[4] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[5] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.

[6] S. Mahadevan and M. Maggioni, "Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes," *Journal of Machine Learning Research*, vol. 8, pp. 2169–2231, 2007.

[7] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K.-R. Müller, Eds. MIT Press, 2000, pp. 1057–1063.

[8] P. Marbach and J. N. Tsitsiklis, "Approximate gradient methods in policy-space optimization of Markov reward processes," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 13, pp. 111–148, 2003.

[9] R. Munos, "Policy gradient in continuous time," *Journal of Machine Learning Research*, vol. 7, pp. 771–791, 2006.

[10] S. Mannor, R. Y. Rubinstein, and Y. Gat, "The cross-entropy method for fast policy search," in *Proceedings 20th International Conference on Machine Learning (ICML-03)*, Washington, US, 21–24 August 2003, pp. 512–519.

[11] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, *Simulation-Based Algorithms for Markov Decision Processes*. Springer, 2007.

[12] S. Whiteson and P. Stone, "Evolutionary function approximation for reinforcement learning," *Journal of Machine Learning Research*, vol. 7, pp. 877–917, 2006.

[13] R. Y. Rubinstein and D. P. Kroese, *The Cross Entropy Method. A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*, ser. Information Science and Statistics, M. Jordan, J. Kleinberg, B. Scholkopf, F. Kelly, and I. Witten, Eds. Springer, 2004.

[14] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM Journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003.

[15] A. Costa, O. D. Jones, and D. Kroese, "Convergence properties of the cross-entropy method for discrete optimization," *Operations Research Letters*, vol. 35, pp. 573–580, 2007.

[16] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Continuous-state reinforcement learning with fuzzy approximation," in *Adaptive Agents and Multi-Agent Systems III*, ser. Lecture Notes in Computer Science, K. Tuyls, A. Nowé, Z. Guessoum, and D. Kudenko, Eds. Springer, 2008, vol. 4865, pp. 27–43.

[17] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings 15th International Conference on Machine Learning (ICML-98)*, Madison, US, 24–27 July 1998, pp. 463–471.