Reaching optimally over the workspace: a machine learning approach

Didier Marin and Olivier Sigaud

***Abstract*—Recent theories of Human Motor Control explain our outstanding coordination capabilities by calling upon an Optimal Control (OC) framework. But OC methods are generally too expensive to be applied on-line and in real-time as would be required to perform everyday movements. An alternative method consists in obtaining a pre-computed feedback policy that performs optimally while being executed reactively. One way to get such a pre-computed policy consists in tuning a parametrized reactive controller so that it converges to optimal behavior.**

In this paper, we demonstrate a method to obtain such a reactive controller that (i) adapts the time of movement based on a compromise between the amount of reward and the effort required to get it, (ii) provides an efficient trajectory from any point to any point in the workspace, (iii) learns from demonstrations of optimal trajectories, (iv) is improving its performance over accumulated experience.

I. Introduction

Optimal Control (OC) is a useful framework for modeling Human Motor Control (HMC) properties [20], [19], [3]. Recently, [13] proposed a model of human reaching movement based on the assumption that HMC is governed by an optimal feedback policy computed at each visited state given a cost function that involves a trade-off between muscular effort and goal-related reward. However, OC methods such as theirs are too expensive to be applied on-line and in real-time as would be required for modeling everyday movements.

Such motor control problems have attracted a lot of attention in the Reinforcement Learning (RL) community these last years (see [12] for an overview). While the RL framework is similar to OC, its methods are designed to improve a parametric feedback controller, called a policy, all along the lifetime of the system through interactions with its environment. Such methods also comes with the benefits of adaptation and can be implemented by using incremental, stochastic optimization methods. From the point of view of RL, motor control problems are part of a broader class of problems which involve continuous state and action spaces and have been addressed in several ways.

A first approach stems from the adaptation of discrete RL techniques to continuous state and action spaces. The core of this approach has been based on Actor-Critic architectures, where an approximation of the expected performance of the policy is updated in parallel with the policy itself. Among these methods, Natural Actor-Critic (NAC) and its episodic variant eNAC [11] have been successfully applied to complex robotics problems [12]. However, these methods have been shown to be very difficult to tune [5] and unstable without adequate features.

Given these difficulties, the attention has shifted towards *direct Policy Search* methods (e.g. [7], [8], [6], [1]), which do not rely on an explicit representation of the expected performance. Instead, they optimize the parameters of a policy using a Stochastic Optimization approach, evaluating the performance through Monte-Carlo sampling. Within this category, *probability-weighted averaging* methods such as the Cross-Entropy Methods (CEMs) [14], CMA-ES [4] and PI^2 [1] are particularly robust, because they do not assume that the objective function is differentiable or even continuous and optimize a *population* of solutions rather than one (such as in gradient descent). CMA-ES can be seen as a variant of the CEM where the updates are "smoothed" over iterations (see [5] for a comparison with NAC). The more recent PI^2 algorithm is very similar to CEM, though it derives from very different first principles [18]. However, it generates an open-loop control rather than a closed-loop one. One can optimize the location of the target state [17], but then the capability to reach a previous target is forgotten.

All these methods are sensitive to local minima. Therefore, the optimization process is generally preceded by a *learning from Demonstration* (LfD) stage that initializes the parametric controller close to a reasonably good local optimum.

In this paper, we evaluate a machine learning approach to get a reactive parametric controller that behaves like the OC model proposed in [13], but performs 20000 times faster. In Section II, we present the OC model of optimally reaching for reward on which our approach is built. Then we present our LfD and stochastic optimization methods. In Section IV, we present the design of the experiments performed to evaluate the properties of the obtained parametric controller. In Section V, empirical results demonstrate the viability of the approach. Finally, Section VI summarizes the results and presents the perspectives of this work.

II. Optimally reaching for reward

The model proposed in [13] is designed to explain how the time of movement emerges from a compromise between the increasing effort required to reach a reward faster and the decreasing subjective value of getting this reward later

Didier Marin (PhD candidate in Robotics) and Olivier Sigaud (Professor in Computer Science) are with:
Université Pierre et Marie Curie
Institut des Systèmes Intelligents et de Robotique - CNRS UMR 7222
Pyramide Tour 55 - Boîte Courrier 173
4 Place Jussieu, 75252 Paris CEDEX 5, France
Contact: firstname.name@isir.upmc.fr

in time. If we consider that the motor system is optimizing the sum of rewards over time minus the cost incurred for reaching them, then a time at which this sum is optimal for a given reward emerges, as illustrated in Fig. 1.

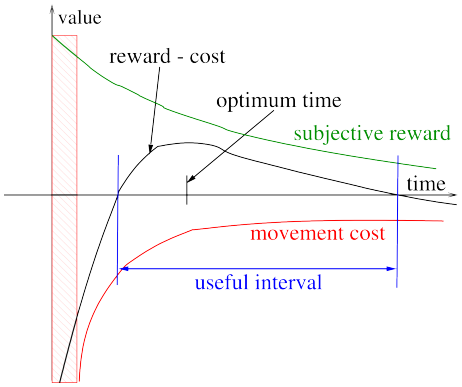

Fig. 1. Optimal movement time. The subjective utility of getting the reward decreases over time (green line). Reaching cannot be performed under a certain time (dashed area) and is less and less costly in terms of efforts as the movement is performed more slowly (red line). However the subjective reward for reaching the goal decreases with time as we are less interested in gains that will occur in a distant future than at the present time. The benefit versus cost criterion, resulting from the sum of the subjective reward and the (negative) cost reaches a maximum for a certain time. When the criterion is negative (outside useful interval), the subject should not move.

The cost function $J(\boldsymbol{u})$ proposed for a control $\boldsymbol{u}$ in [13] is

$$J(\boldsymbol{u}) = \int_0^\infty e^{-t/\alpha}[\rho R(\boldsymbol{x}_t) - \nu L(\boldsymbol{u}_t)]dt \qquad (1)$$

where ρ is the weight of the reward term and ν the weight of the effort term. $R(\boldsymbol{x}_t)$ is the immediate reward function that equals 1 at the target state and is null everywhere else. $L(\boldsymbol{u}_t)$ is the movement cost function, we take $L(\boldsymbol{u}_t) = \|\boldsymbol{u}_t\|^2$ as in many HMC models. The continuous-time discount factor α account for the "greediness" of the controller.

The corresponding near optimal deterministic policy is obtained through a computationally expensive variation calculus method. The feedback controller, resulting from the coupling of this policy with an optimal state estimator, drives the plant towards the rewarded state. Given that the policy does not take the presence of noise in the model into account, the actions must be computed again at each time step depending on the new state reached by the plant. Overall, generating a trajectory with this method is extremely costly.

Hereafter, this controller is called Near-Optimal Planning System (NOPS). Indeed, the trajectories are not optimal in the strict sense, given the presence of non-modeled noise. A crucial feature of this model is that the generated movements do not depend on time, as opposed to state-of-the-art OC algorithms such as iLQG [21]. This results in the possibility to learn stationary policies from the model. In our framework, such policies are learned from demonstration with XCSF.

III. Machine Learning Methods

A. Learning from Demonstration with XCSF

Learning Classifier Systems (LCSs) is a Machine Learning family of rule-based systems [15]. The XCS [22] is an efficient accuracy-based LCS designed to solve classification problems and sequential decision problems. XCSF [23] is an evolution of XCS towards function approximation.

As any LCS, XCSF manages a population of rules, called *classifiers*. These classifiers contain a condition part and a prediction part. In XCSF, the condition part defines the region of validity of a local model whereas the prediction part contains the local model itself. XCSF is a generic framework that can use different kinds of prediction models (linear, quadratic, etc.) and can pave the input space with different families of regions (Gaussian, hyper-rectangular, etc.). In the context of this paper, we only consider the case of linear prediction models and Gaussian regions.

A classifier defines a domain $\phi_i(\boldsymbol{z})$ and uses a corresponding linear model β_i to predict a local output vector $\boldsymbol{y}_i$ relative to an input vector $\boldsymbol{x}_i$. The linear model is updated using the Recursive Least Squares (RLS) algorithm, the incremental version of the Least Squares method.

The classifiers in XCSF form a population P that clusters the condition space into a set of overlapping prediction models. XCSF uses only a subset of the classifiers to generate an approximation. Indeed, at each iteration, XCSF generates a match set M that contains all reliable classifiers in the population P whose condition space $\mathcal{Z}$ matches the input data $\boldsymbol{z}$ i.e., for which $\phi_i(\boldsymbol{z})$ is above a threshold ϕ_0.

In XCSF, the output $\hat{\boldsymbol{y}}$ is given for a $(\boldsymbol{x}, \boldsymbol{z})$ pair as the sum of the linear models of each matching classifier i weighted by its fitness F_i

$$\hat{\boldsymbol{y}}\left(\boldsymbol{x}, \boldsymbol{z}\right) = \frac{1}{F\left(\boldsymbol{z}\right)} \sum_{i=1}^{n_M} F_i\left(\boldsymbol{z}\right) \hat{\boldsymbol{y}}_i\left(\boldsymbol{x}\right) \qquad (2)$$

where $F\left(\boldsymbol{z}\right) = \sum_{i=1}^{n_M} F_i\left(\boldsymbol{z}\right)$ and n_M is the number of classifiers in the match set M. In all other respects, the mechanisms that drive the evolution of P are directly inherited from XCS. In sum, XCSF is designed to evolve maximally accurate approximations of the learned function. A more complete description of XCSF can be found in [2].

In our architecture, XCSF is used to learn a policy from demonstration. More precisely, a set of near-optimal state-action trajectories generated by the NOPS provides supervised learning samples, using the state of the plant as the condition and prediction space input and the action as the output on which regression is performed. By feeding XCSF with such samples, we generate an action for any state within the range of the population of classifiers. Using a default action $\boldsymbol{a}_{default}$ for states that are not covered by the population (that is for which XCSF does not predict anything), we get a mapping from states to actions i.e., a deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. We call it the "XCSF policy" and note it $\pi_{\boldsymbol{\theta}_0}$. In a second step, it is improved by

a direct policy search method called Cross-Entropy Policy Search (CEPS).

B. Improving the parametric controller with CEPS

The XCSF policy $\pi_{\boldsymbol{\theta}_0}$ is parametric since each classifier has parameters in its condition and prediction parts. To optimize a parametric policy $\pi_{\boldsymbol{\theta}}$ with respect to its performance $J(\boldsymbol{\theta})$, we developed a direct Policy Search method based on the CEM.

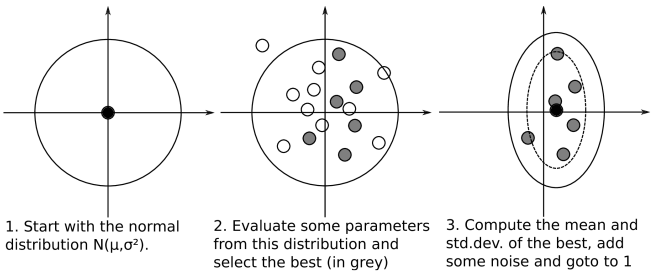

Fig. 2. Schematic view of the Cross-Entropy method.

The general CEM is given in Alg. 1 and illustrated in Fig. 2. For computational complexity reasons, our implementation of CEM only updates the diagonal of the covariance matrix. A CEM can be applied straightforwardly to optimization of policy parameters $\boldsymbol{\theta}$, which results in the CEPS method, described in Alg. 2. $\pi_{\boldsymbol{\theta}_0}$ is adapted using CEPS where J is the cost over trajectories (see 1). Each resulting policy $\pi_{\boldsymbol{\theta}}$ corresponds to a dot (i.e. a CEPS sample) in Fig. 2. In practice, $\boldsymbol{\theta}$ only contains the weights of each local model β_i.

Algorithm 1 CEM_iter

Require: $\{(\boldsymbol{\theta}_{(i)}, J_{(i)})\}_{i=0\cdots N}$: set of N solution-value pairs
ρ: proportion of the best solutions to use for the update
σ_n^2: additional noise term

Sort the $\boldsymbol{\theta}_{(i)}$ according to $J_{(i)}$
Compute the set $\mathcal{S}_\rho$ of the $\max(1, N \times \rho)$ best solutions
$\boldsymbol{\mu} \leftarrow \text{mean}(\mathcal{S}_\rho)$
$\boldsymbol{\sigma}^2 \leftarrow \text{std.dev}(\mathcal{S}_\rho) + \sigma_n^2$
return mean $\boldsymbol{\mu}$ and std.dev. $\boldsymbol{\sigma}^2$

IV. Experimental design

We now illustrate the use of XCSF and CEPS for learning to control an arm to reach diverse targets with its end-effector. The plant is a simulated two degrees-of-freedom planar arm controlled by 6 muscles. The equations of the model and the specification of the control problem are given in [9].

A. Arm model set-up

We choose a point $\mathcal{P}$ that correspond to $\boldsymbol{q} = [0.5, 0.59]$, located in the upper-right part of the reachable space (see Fig. 3), which is used through all experiments either as starting point or target. Note that we display the plant

Algorithm 2 Cross-Entropy Policy Search (CEPS)

Require: $\{\boldsymbol{s}^*_{(j)}\}_{j=1\cdots M}$: set of target states
$(\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0^2)$: initial mean and std.dev of the $\boldsymbol{\theta}$ distribution
ρ: proportion of the best samples to use for the update
σ_n^2: additional noise term
N: number of sample policies to draw
K: number of iterations

for $k = 1 \cdots K$ **do**
  for $i = 1 \cdots N$ **do**
    Draw a sample $\boldsymbol{\theta}_{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2)$
    for $j = 1 \cdots M$ **do**
      Perform an episode τ_j for target $\boldsymbol{s}^*_{(j)}$ following $\pi_{\boldsymbol{\theta}_{(i)}}$
    end for
    Compute the global performance of $\pi_{\boldsymbol{\theta}_{(i)}}$:
    $J_{(i)} = \frac{1}{M}\sum_{j=1}^{M}\sum_{t=0}^{|\tau_j|-1}\gamma^t r_{j,t}$ *where $r_{j,t}$ is the reward at time t for the episode τ_j*
  end for
  Perform a CEM iteration (Alg.1):
  $\boldsymbol{\mu}_{k+1}, \boldsymbol{\sigma}^2_{k+1} = CEM\_iter(\{(\boldsymbol{\theta}_{(i)}, J_{(i)})\}_{i=0\cdots N}, \rho, \sigma_n^2)$
end for
return optimized $\boldsymbol{\theta} = \boldsymbol{\mu}_{K+1}$

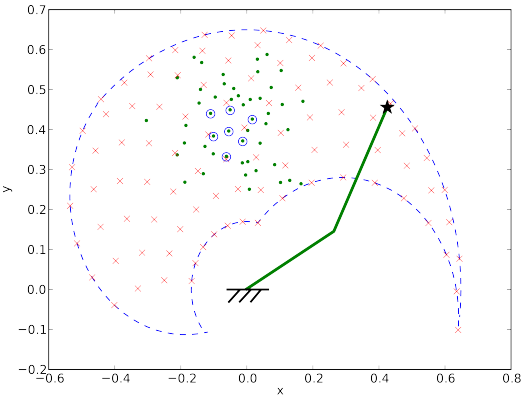

Fig. 3. The arm workspace. The reachable space is delimited by a spiral-shaped envelope. The two segments of the arm are represented by two bold green lines. Point $\mathcal{P}$ is represented as a star, $\mathcal{S}$ points as green dots, $\mathcal{L}$ points as red crosses and $\mathcal{T}$ as blue circles.

in the operational space (x, y) although our state space representation does only include articular coordinates.

Reaching is successful when the Euclidean distance from the end-effector to the target, i.e. $\sqrt{(x^* - x)^2 + (y^* - y)^2}$, is below $0.01m$ and the absolute end-effector speed, i.e. $\sqrt{\dot{x}^2 + \dot{y}^2}$, is below $0.1m.s^{-1}$. These thresholds are large enough for allowing some exploration without the risk of "losing track" of the target, which may occur in the learning context. For NOPS, such thresholds are not necessary since it is capable of arbitrary precision. The cost function is parametrized such that the continuous-time formulation used by NOPS (see (1)) is equivalent to the discrete-time formulation used by CEPS, i.e., $J(\boldsymbol{u}) = \sum_t \gamma^t[\zeta R(\boldsymbol{x}_t) - \kappa L(\boldsymbol{u}_t)]$. The continuous-time parameters are set to $\alpha = 1$, $\rho = 200$

and $\nu = 40$, which discrete-time counterparts are $\gamma = \alpha e^{-\Delta t}$, $\zeta = \nu \Delta t$ and $\kappa = \rho\gamma$, where Δt is the time step of the simulation.

B. Experiments

We perform three experiments. For the first two, we create two sets of articular positions, a small set $\mathcal{S}$ and a large set $\mathcal{L}$. $\mathcal{S}$ contains 50 targets drawn according to a normal distribution centered on $x = -0.059$ and $y = 0.44$ and with standard deviation 0.1^2. $\mathcal{L}$ set contains 94 targets from a 10×10 grid over the articular space (NOPS diverged for 6 of them, which were excluded). For the last experiment we create a set $\mathcal{T}$ of 7 positions located near the center of $\mathcal{S}$.

1) Learning a controller to many targets with XCSF: In the first experiment, the arm always starts from $\mathcal{P}$ and target positions are picked from $\mathcal{S}$ and $\mathcal{L}$. First, the NOPS is used to generate one trajectory for each target in $\mathcal{S}$. Then, XCSF learns from the trajectories generated by the NOPS for targets in $\mathcal{S}$, and is tested as a policy on both $\mathcal{S}$ and $\mathcal{T}$ as sets of targets. The condition and prediction space contain states $\boldsymbol{s}$ as defined in Section IV-A. The performance and the trajectories obtained with $\pi_{\boldsymbol{\theta}_0}$ are compared to those generated by the NOPS, to see how good $\pi_{\boldsymbol{\theta}_0}$ generalizes to other targets.

2) Learning a controller from many start states with XCSF: In the second experiment, the start and target positions are reversed: the arm starts from positions in $\mathcal{S}$ and $\mathcal{L}$ and $\mathcal{P}$ is used as a common target. The rest of the protocol is similar to the first experiment: the NOPS is used to generate trajectories from points of $\mathcal{S}$ to $\mathcal{P}$ and XCSF then learns $\pi_{\boldsymbol{\theta}_0}$ from these trajectories. $\pi_{\boldsymbol{\theta}_0}$ is tested using points in both $\mathcal{S}$ and $\mathcal{L}$ as start positions. The rest is as above.

3) Effect of goal reward on movement time: In order to study the effect of the reward on the movement time, we design a new controller where the weight of the reward term ν is given as a new dimension of the input space. In other words, the controller knows from the start how interesting the current target is and this interest is varied. We first train the controller from $\mathcal{P}$ to targets in $\mathcal{T}$ with 6 different values of ν, ranging from 1 to 100, using the same LfD methodology as in Section IV-B1. Then we focus on the central target among $\mathcal{T}$ (see Fig. 3) and train again the controller for more ν values using CEPS. Once this controller is trained, we measure its movement time for each ν value and compare it to the time obtained with the NOPS.

C. Experimental set-up and Parameters

We use the JavaXCSF [16] implementation of XCSF, and all algorithms are implemented in Java. Computation times are measured on an Intel Core 2 Duo E8400 @ 3 GHz with 4 GB RAM. XCSF is tuned as follows. The number of iterations is set to $1,000,000$ and the maximum size of the population to 500. The input are normalized: the target and current positions are bounded by the reachable space and the speed is bounded by $[-100, +100]\ rad.s^{-1}$. The default action $\boldsymbol{a}_{default}$ is set to a vector of zeros i.e., no muscular activation. After tuning empirically the parameters, the learning rate α (named `beta` in JavaXCSF) is set to 1.0, and compaction is disabled. For both CEPS variants, the number of iterations K is such that the method runs up to a time limit of 2 hours. Each iteration consists of the evaluation of $N = 100$ policies. The proportion of selected episodes ρ is set to 0.8, i.e., the $N \times \rho = 80$ best policies are used for the update. The initial variance $\boldsymbol{\sigma}^2$ is set to 0.1 and the additional noise $\sigma_n^2 = 10^{-3}$.

TABLE I
MEAN AND STANDARD DEVIATION OF THE PERFORMANCE OVER TARGETS IN EXPERIMENT 1. FOR XCSF AND CEPS, THE PERFORMANCE OF THE SET USED FOR LEARNING ARE IN BOLD.

start / target	$\mathcal{P}/\mathcal{S}$	$\mathcal{P}/\mathcal{L}$
NOPS	28.22 ± 2.06	27.46 ± 3.73
XCSF	$\mathbf{27.45 \pm 2.35}$	2.44 ± 46.03
CEPS	22.96 ± 11.50	$\mathbf{12.76 \pm 22.28}$

TABLE II
MEAN AND STANDARD DEVIATION OF THE PERFORMANCE OVER START POSITIONS IN EXPERIMENT 2. FOR XCSF AND CEPS, THE PERFORMANCE OF THE SET USED FOR LEARNING ARE IN BOLD.

start / target	$\mathcal{S}/\mathcal{P}$	$\mathcal{L}/\mathcal{P}$
NOPS	27.68 ± 2.16	26.37 ± 5.11
XCSF	$\mathbf{27.51 \pm 2.67}$	3.99 ± 28.84
CEPS	18.98 ± 11.84	$\mathbf{16.39 \pm 16.36}$

V. RESULTS

In this section, we first study whether the policy learned with XCSF is similar to the one obtained with the NOPS for the same targets, how well XCSF generalizes over different targets and starting points, and whether CEPS is able to improve the performance of the learned policy. Then we evaluate the ability of XCSF and CEPS to reproduce the characteristic movement time depending on the value of the goal reward-related parameter ν.

A. Performance of XCSF policy

The average running time to get one trajectory from the NOPS is ~ 10 minutes. From $\pi_{\boldsymbol{\theta}_0}$, it is ~ 30 milliseconds.

1) Single start position to multiple targets: Fig. 4(a) shows the performance of the NOPS using $\mathcal{P}$ as the start position and as a function of the target position, obtained by interpolating the performance of the NOPS trajectories for target positions in $\mathcal{L}$. Fig. 4(b) shows the performance of one representative $\pi_{\boldsymbol{\theta}_0}$, which is very close to the NOPS performance for targets located near the learning set $\mathcal{S}$. One can see that the relative performance decreases with the distance to the learning set.

Fig. 4(c) shows the performance of $\pi_{\boldsymbol{\theta}}$ after applying CEPS to optimize its parameters. One can see that the performance is globally improved over the whole reachable space. In particular, the region around $(x = -0.3, y = 0)$, where the performance of $\pi_{\boldsymbol{\theta}_0}$ was very poor, is improved a lot. The results in Table I confirm quantitatively this visual feeling.

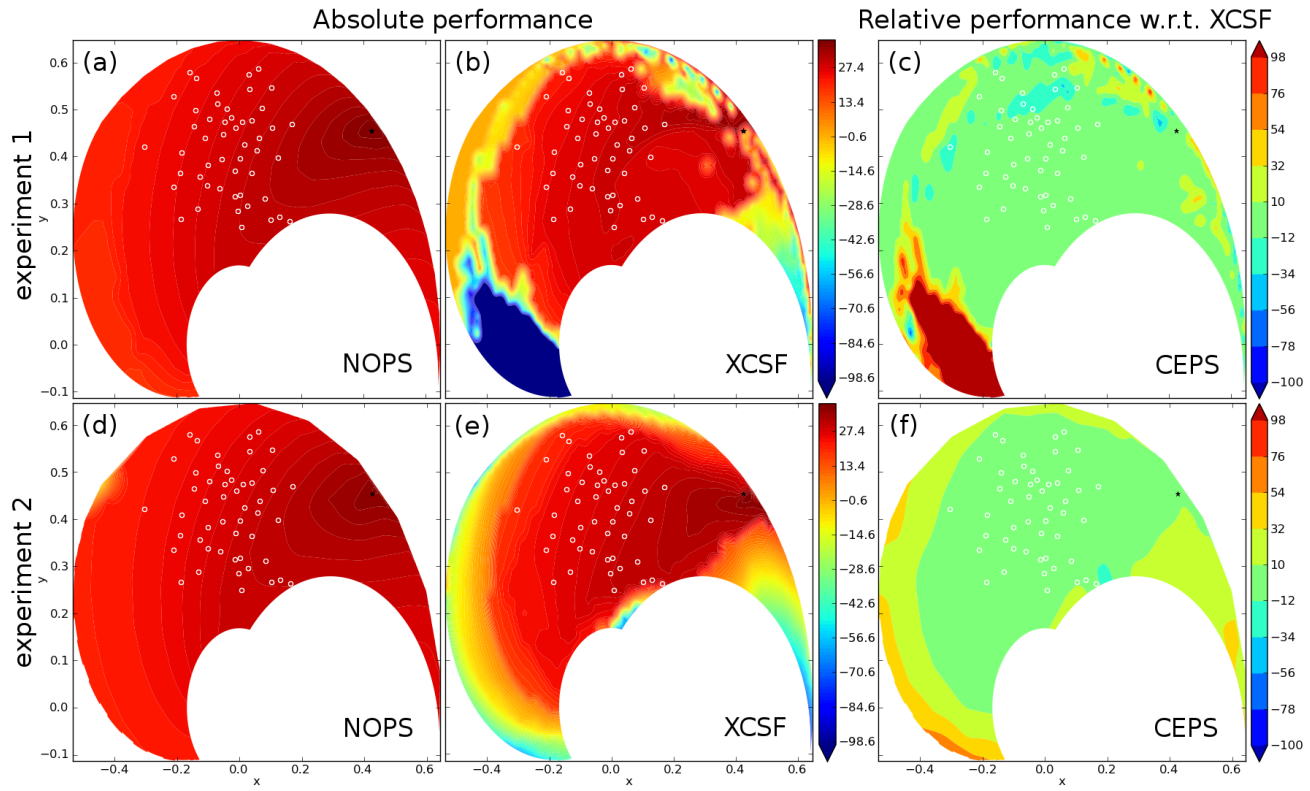

Fig. 4. The first column represents the absolute performance for fixed start position $\mathcal{P}$ and given the target position when using (a) the NOPS and (b) $\pi_{\boldsymbol{\theta}_0}$, and (c) relative performance of $\pi_{\boldsymbol{\theta}}$ optimized by CEPS, with respect to $\pi_{\boldsymbol{\theta}_0}$. The second column correspond to the second experiment, where the target position is always $\mathcal{P}$ and the performance is a function of the start position. These plot are obtained by interpolating the performances for a large grid of target (first line) or start (second line) positions. The set $\mathcal{S}$ is represented by white circles, and $\mathcal{P}$ by a star. Performance is computed according to (1).

2) Multiple start positions to single target: Fig. 4(d) shows the performance of the NOPS using $\mathcal{P}$ as the target position and as a function of the start position, obtained by interpolating the performance of the NOPS trajectories for start positions in $\mathcal{L}$. Fig. 4(e) shows the performance of one representative $\pi_{\boldsymbol{\theta}_0}$, which is very close to the NOPS performance when the trajectory starts close to the learning set $\mathcal{S}$. One can see that the generalization performance is much more regular in this experiment than in the previous (Fig. 4(b)). Fig. 4(f) shows the performance of $\pi_{\boldsymbol{\theta}}$ after optimization of $\boldsymbol{\theta}$ by CEPS. Once again, the performance is globally improved over the whole reachable space. The bottom-left and bottom-right regions, which were the worst regions in Fig. 4(e), are improved significantly. From this visual feeling and the results in Table II, it seems fairly easier to learn how to reach a single target from multiple start positions.

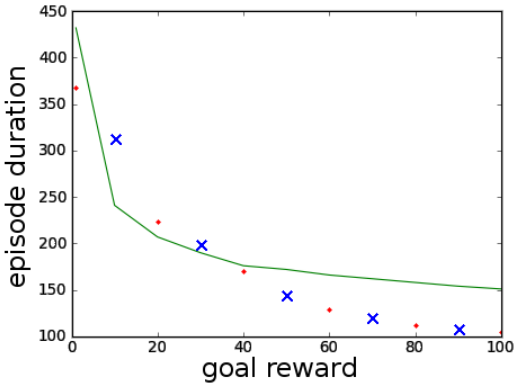

Fig. 5. Movement time as a function of the goal reward ν for the central target of $\mathcal{T}$ for the NOPS (green line) and for an $\pi_{\boldsymbol{\theta}_0}$ (red dots and blue crosses). The red dots represents the ν values used for learning by demonstration and the blue crosses the additional values used to evaluate generalization.

B. *Effect of reward on time*

Fig. 5 shows the movement time as a function of the goal-related parameter ν, following the experimental procedure described in Section IV-B3. XCSF is trained using NOPS trajectories for each point of $\mathcal{T}$ as the target and for ν being either 1,20,40,60,80 or 100. It is then tested for the central point of $\mathcal{T}$ with intermediate ν values $1, 10, 20, \cdots, 90, 100$. One can see that $\pi_{\boldsymbol{\theta}_0}$ qualitatively reproduces the general results obtained with the NOPS. Furthermore, $\pi_{\boldsymbol{\theta}_0}$ is able to generalize well for other values of ν, as demonstrated by the good interpolation of movement times for intermediate values. However, further improvement with CEPS did not seem to bring the movement time closer to those of the NOPS. Further experiments will be necessary to investigate this problem.

VI. Conclusion

In this paper, we have proposed a method to learn and optimize a parametric controller that can store and generate in real-time near-optimal trajectories to a set of targets and reward values in the workspace of the system. As opposed to OC methods, our method does not need constant replanning, which makes our controller about 20000 times faster than the method presented in [13]. Moreover, the controller representation based on XCSF gives good generalization capabilities which makes the learning process easier. We have shown that using a stochastic optimization algorithm could improve the policy over the accumulated experience of the agent, particularly in the areas where the performance resulting from LfD is far from optimal. Our CEPS algorithm benefits from the robustness of probability-weighted averaging methods and can adapt the controller to changes in the environment, such as the presence of a force field. From a neurosciences perspective, our parametric controller approach might be seen as a computational model of how the Central Nervous System might store the capability to reach optimally and in real-time from any state and to any target.

However, from our empirical results, one can see that the obtained performance after a limited stochastic optimization period is still not close enough to that of the OC method. In particular, generalization and performance improvement are still limited with respect to the reward value. To overcome these limitations, we may consider a *lifelong learning* approach, where the system is optimizing its controller parameters throughout his life-time, yielding the slow convergence to a better optimum. Another possibility consists in using more sophisticated alternatives to the CEM on which CEPS is based, such as CMA-ES or PI^2.

Finally, we cannot perform further LfD after optimizing the policy using CEPS. We intend to include in our framework some active learning capabilities that would allow the system to choose the trajectories from which its policy improvement would be the greatest, but also to chose between LfD and CEPS based on its current performance and on the computational cost of both learning methods.

We envision applying our approach to an assistance robot. We are designing a robot that will help motor-impaired patients to transfer from a seated position to a standing position using a bio-mechanical criterion based on efforts and stability. With the methods presented here, we will adapt in real-time the controller of the robot on-line to specific patients [10]. On the long term, we would like to apply these techniques to the whole body control of humanoid robots like iCub, so that they perform more human-like movements.

Acknowledgments

This work was supported by the Ambient Assisted Living Joint Programme of the European Union and the National Innovation Office (DOMEO-AAL-2008-1-159), more at http://www.aal-domeo.eu.

References

[1] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal. Learning variable impedance control. *The International Journal of Robotics Research*, 30(7):820, 2011.

[2] M. V. Butz and O. Herbort. Context-dependent predictions and cognitive arm control with XCSF. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1357–1364. ACM New York, NY, USA, 2008.

[3] E. Guigon, P. Baraduc, and M. Desmurget. Optimality, stochasticity and variability in motor behavior. *Journal of Computational Neuroscience*, 24(1):57–68, 2008.

[4] N. Hansen, S. Muller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[5] V. Heidrich-Meisner and C. Igel. Similarities and differences between policy gradient methods and evolution strategies. In *Proceedings of the 16th ESANN*, 2008.

[6] S. Ivaldi, M. Fumagalli, F. Nori, M. Baglietto, G. Metta, and G. Sandini. Approximate optimal control for reaching and trajectory planning in a humanoid robot. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1290–1296. IEEE, 2010.

[7] J. Kober and J. Peters. Policy search for motor primitives in robotics. *NIPS*, pages 1–8, 2008.

[8] P. Kormushev, S. Calinon, and D. G. Caldwell. Robot Motor Skill Coordination with EM-based Reinforcement Learning. In *Proc. IROS*, 2010.

[9] D. Marin, J. Decock, L. Rigoux, and O. Sigaud. Learning cost-efficient control policies with xcsf: generalization capabilities and further improvement. In *Proceedings of the 13th annual Conference on Genetic and Evolutionary Computation*, pages 1235–1242. ACM, 2011.

[10] V. Pasqui, L. Saint-Bauzel, and O. Sigaud. Characterization of a least effort user-centered trajectory for sit-to-stand assistance user-centered trajectory for sit-to-stand assistance. In *Proceedings IUTAM*. IUTAM, june 2010.

[11] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, Nov., 2007.

[12] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks : the official journal of the International Neural Network Society*, 21(4):682–97, 2008.

[13] L. Rigoux, O. Sigaud, A. Terekhov, and E. Guigon. Movement duration as an emergent property of reward directed motor control. In *Proceedings of the Annual Symposium Advances in Computational Motor Control*, 2010.

[14] R. Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

[15] O. Sigaud and S. Wilson. Learning classifier systems: a survey. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 11(11):1065–1078, 2007.

[16] P. O. Stalph and M. V. Butz. Documentation of JavaXCSF. Technical report, COBOSLAB, 2009.

[17] F. Stulp and S. Schaal. Hierarchical reinforcement learning with movement primitives. In *IEEE-RAS International Conference on Humanoid Robots*, pages 231–238. IEEE, 2011.

[18] E. Theodorou, J. Buchli, and S. Schaal. Reinforcement learning of motor skills in high dimensions: a path integral approach. In *International Conference on Robotics and Automation*, pages 2397–2403. IEEE, 2010.

[19] E. Todorov. Optimality principles in sensorimotor control. *Nature Neurosciences*, 7(9):907–915, 2004.

[20] E. Todorov and M. I. Jordan. Optimal feedback control as a theory of motor coordination. *Nature Neurosciences*, 5(11):1226–1235, 2002.

[21] E. Todorov and W. Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the American Control Conference*, pages 300–306, 2005.

[22] S. W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[23] S. W. Wilson. Function approximation with a classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 974–981, San Francisco, California, USA, 2001. Morgan Kaufmann.