Towards fast and adaptive optimal control policies for robots: A direct policy search approach

Didier Marin and Olivier Sigaud

***Abstract*—Optimal control methods are generally too expensive to be applied on-line and in real-time to the control of robots. An alternative method consists in tuning a parametrized reactive controller so that it converges to optimal behavior. In this paper we present such a method based on the “*direct Policy Search*” paradigm to get a cost-efficient control policy for a simulated two degrees-of-freedom planar arm actuated by six muscles. We learn a parametric controller from demonstration using a few near-optimal trajectories. Then we tune the parameters of this controller using two versions of a *Cross-Entropy Policy Search* method that we compare. Finally, we show that the resulting controller is 20000 times faster than an optimal control method producing the same trajectories.**

I. Introduction

The mission of robots is changing deeply. In contrast with factory robots always repeating the same task in a perfectly known environment, service robots will have to deal with complex tasks in the presence of humans, while being subject to many unforeseen perturbations. As a result of this evolution, a whole set of new control principles is needed. Among such principles, the study of Human Motor Control (HMC) suggests to call upon optimality and adaptation. However, Optimal Control (OC) methods are generally too expensive to be applied on-line and in real-time to the control of robots.

A straightforward solution to this cost problem that also comes with the benefits of continuous adaptation consists in calling upon incremental, stochastic optimization methods to improve the behavior of the system all along its lifetime through its interactions with its environment. This generates intensive research and, in particular, the *direct Policy Search* methods are providing more and more interesting results (e.g. [7], [8], [6], [1]). Since such methods are generally sensitive to local minima problems, the optimization process is generally preceded by a *Learning from Demonstration* (LfD) stage that drives the process to the preferred local optimum.

In [9], the potential of such an approach was demonstrated through a simple reaching experiment with a 2D arm controlled by 6 muscles. An OC method based on a computationally expensive variational calculus process was replaced by the combination of two adaptive components. First, a parametric controller was learned from demonstration using a few near-optimal trajectories, but the resulting policy was suboptimal. Thus, in a second step, the parametric controller was improved by a stochastic optimization process acting on the parameters of the controller.

However, the empirical study presented in [9] showed that, despite a good generalization capability and global improvement in performance, the stochastic optimization process was detrimental to performance in the area where the initial controller learned from demonstration was already nearly optimal. In this paper, we focus on this specific issue. We show that the global optimization can be replaced by a more local approach that only acts where the performance must be improved.

The paper is organized as follows. In Section II, we present our LfD and stochastic optimization methods, together with some related work. In Section III, we present the design of the experiments. Results of the comparison between the global method and its local counterpart are given in Section IV. Finally, Section V summarizes the results and presents the perspectives of this work.

II. Methods

In this section, we describe the methods used in our work, summarized in Fig. 1.

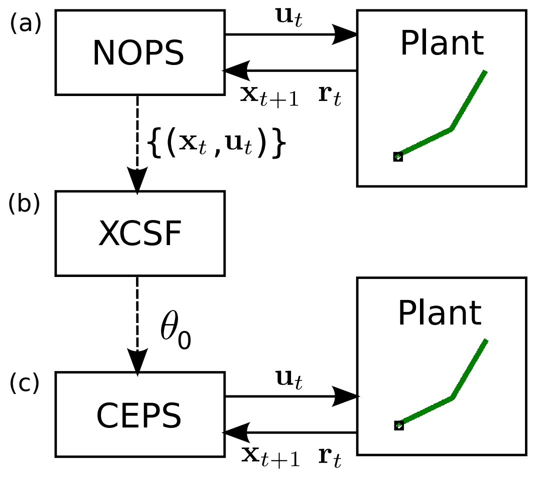
(a)
NOPS
$\mathbf{u}_t$
Plant
$\mathbf{x}_{t+1}$ $\mathbf{r}_t$
$\{(\mathbf{x}_t, \mathbf{u}_t)\}$
(b)
XCSF
θ_0
(c)
CEPS
$\mathbf{u}_t$
Plant
$\mathbf{x}_{t+1}$ $\mathbf{r}_t$

Fig. 1. General architecture of the experiments. The nature and role of each box is detailed in the text.

A. Generating near optimal control with the NOPS

In [12], the authors proposed a model of human reaching movements based on the OC paradigm. The model is based on

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`

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

$$r(\boldsymbol{s}_t, \boldsymbol{a}_t) = \alpha\|\boldsymbol{a}_t\|^2 - \beta g(\boldsymbol{s}_t) \qquad (1)$$

where α is a weight on the effort term, g is a function that equals 1 at a target state $\boldsymbol{s}^*$ and is null everywhere else, and β is a weight on the reward term.

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. This results in the possibility to learn stationary policies from the model. In our framework, such policies are learned from demonstration with XCSF.

B. Learning from demonstration with XCSF

Learning Classifier Systems (LCSs) is a Machine Learning family of rule-based systems [16]. The XCS [2], [20] is an efficient accuracy-based LCS designed to solve classification problems and sequential decision problems. XCSF [21], [22] 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[1].

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

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

where $F(\boldsymbol{z}) = \sum_{i=1}^{n_M} F_i(\boldsymbol{z})$ 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 [3], [4].

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 (Fig. 1 (b)), 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. 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).

C. Improving the XCSF policy with CEPS

The XCSF policy $\pi_{\boldsymbol{\theta}_0}$ is parametric since each classifier has parameters in its condition and prediction parts. A standard way to optimize a parametric policy $\pi_{\boldsymbol{\theta}}$ given an objective function $J(\boldsymbol{\theta})$ consists in performing a gradient descent over $\boldsymbol{\theta}$ giving rise to *Gradient Policy Search* methods. This class of methods has attracted a lot of attention in Reinforcement Learning (see [11] for an overview). However, despite convincing results in robotics [7], [8], such methods are complex and sensitive to many hyper-parameters. An alternative family that gives better results comes from *probability-weighted averaging* methods such as CMA-ES, Cross-Entropy and PI^2 [1].

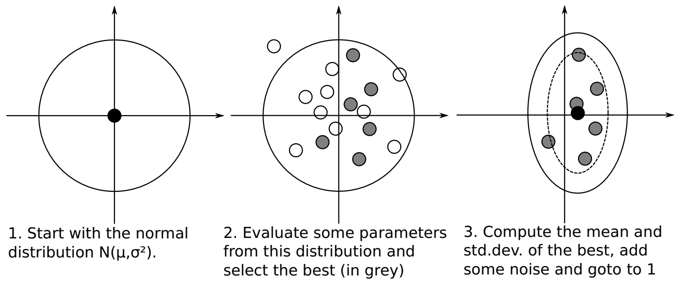
1. Start with the normal distribution N(μ,σ²).
2. Evaluate some parameters from this distribution and select the best (in grey)
3. Compute the mean and std.dev. of the best, add some noise and goto to 1

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

By contrast with Gradient Policy Search methods, Cross-Entropy Methods (CEMs) [13], [14] do not assume that the objective function is differentiable or even continuous. CMA-ES is a more general evolutionary framework that contains CEMs as a special case [5]. More recently, PI^2 has been

[1]This threshold is named θ_m in [3]

proposed as another probability-weighted averaging method that is very similar to CEMs though it derives from very different first principles [19]. The general CEM is given in Alg. 1 and illustrated in Fig. 2. For more details, see [9].

A CEM can be applied straightforwardly to Policy Search, using the policy parameters $\boldsymbol{\theta}$ as solutions and a performance criterion J over targets as objective function. This results in the CEPS method [9]. $\pi_{\boldsymbol{\theta}_0}$ is adapted using CEPS (Fig. 1 (c)) where J is a discounted sum of costs $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ 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
  $\rho$: proportion of the best solutions to use for the update
  $\sigma^2_{noise}$: 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
  $\boldsymbol{\mu} \leftarrow$ mean($\mathcal{S}_\rho$)
  $\boldsymbol{\sigma}^2 \leftarrow$ std.dev($\mathcal{S}_\rho$) $+ \sigma^2_{noise}$
  return mean $\boldsymbol{\mu}$ and std.dev. $\boldsymbol{\sigma}^2$
```

Algorithm 2 "Global" Cross-Entropy Policy Search (GCEPS)

```
Require: $\{\boldsymbol{s}^*_{(j)}\}_{j=1\cdots M}$: set of target states
  $(\boldsymbol{\mu}_0, \boldsymbol{\sigma}^2_0)$: initial mean and std.dev of the $\boldsymbol{\theta}$ distribution
  $\rho$: proportion of the best samples to use for the update
  $\sigma^2_{noise}$: 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 $\boldsymbol{\theta}_{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\sigma}^2_k)$
      for $j = 1 \cdots M$ do
        Perform an episode $\tau_j$ 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 $\tau_j$
    end for
    Perform a CEM iteration (Alg.1):
    $\boldsymbol{\mu}_{k+1}, \boldsymbol{\sigma}^2_{k+1} = CEM\_iter(\{(\boldsymbol{\theta}_{(i)}, J_{(i)})\}_{i=0\cdots N}, \rho, \sigma^2_{noise})$
  end for
  return optimized $\boldsymbol{\theta} = \boldsymbol{\mu}_{K+1}$
```

D. From global CEPS to local CEPS

In [9], CEPS was optimizing a global criterion: the mean performance over all targets. This "global" CEPS (Alg. 2), called GCEPS hereafter, induced a local loss of performance in the demonstration region where $\pi_{\boldsymbol{\theta}_0}$ was already good. We concluded that, to circumvent this effect, we should train $\pi_{\boldsymbol{\theta}_0}$ on one target at a time using a more local criterion

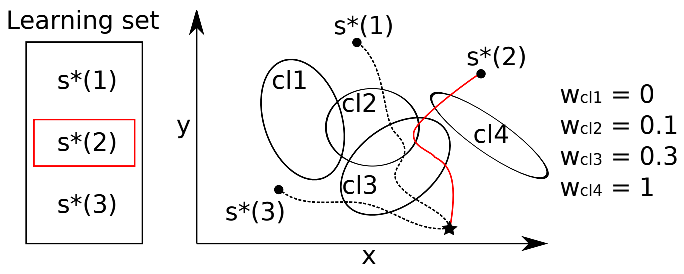

Fig. 4. Schema of the local policy parameters weighting process. The selected target is $\boldsymbol{s}^* = \boldsymbol{s}^*_{(k)}$ with $k = 2$ here. The central figure represents a two-dimensional state-space with some XCSF classifiers $cl_i, i \in \{1, \cdots, 4\}$. The weights w are computed as follows: first, for each target $\boldsymbol{s}^*_{(j)}$ in the learning set (left part), we perform an episode and get a state-space trajectory τ_j. Then, for each trajectory, LCEPS computes $n_{i,j}$ the number of times the classifier cl_i matches a state from the trajectory τ_j. Finally, the weight for all parameters associated which a classifier cl, noted w_{cl} (right part), is computed such that $w_{cl_i} = 1$ if $n_{i,k} > 0$, 0 else.

that would improve the performance only with respect to the current target. This local method should act preferentially on the classifiers that most need it, without disrupting P.

Thus, we propose here a "local" variant of CEPS (Alg. 3), called LCEPS hereafter, which is implemented as follows: each iteration begins by choosing a target state $\boldsymbol{s}^*$ for which a local update is performed using the CEM.

Algorithm 3 "Local" Cross-Entropy Policy Search (LCEPS)

```
Require: as in the global case
  for $k = 1 \cdots K$ do
    Choose a target $\boldsymbol{s}^*$ from the set $\{\boldsymbol{s}^*_{(j)}\}_{j=1\cdots M}$
    Perform an episode $\tau$ for target $\boldsymbol{s}^*$ following $\pi_{\boldsymbol{\mu}_k}$
    Compute weights $w[j] \in [0,1]$ for each parameter $j$
    based on their relevance during episode $\tau$
    Compute the std.dev. $\bar{\boldsymbol{\sigma}}^2$ such that $\bar{\boldsymbol{\sigma}}^2[j] = \boldsymbol{\sigma}_k$ if $w[j] > 0$
    and 0 otherwise
    for $i = 1 \cdots N$ do
      Draw a sample $\boldsymbol{\theta}_{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_k, \bar{\boldsymbol{\sigma}}^2)$
      Perform an episode $\tau_{(i)}$ for target $\boldsymbol{s}^*$ following $\pi_{\boldsymbol{\theta}_{(i)}}$
      Compute the local performance of $\pi_{\boldsymbol{\theta}_{(i)}}$:
      $J_{(i)} = \sum_{t=0}^{|\tau_{(i)}|-1}\gamma^t r_t$ where $r_t$ is the reward at time $t$
      for episode $\tau_{(i)}$
    end for
    Perform a CEM iteration (Alg.1):
    $\boldsymbol{\mu}', \bar{\boldsymbol{\sigma}}'^2 = CEM\_iter(\{(\boldsymbol{\theta}_{(i)}, J_{(i)})\}_{i=0\cdots N}, \rho, \sigma^2_{noise})$
    Update the $\boldsymbol{\theta}$ distribution using the local CEM result
    $\boldsymbol{\mu}_{k+1} = \boldsymbol{w} \times \boldsymbol{\mu}' + (1 - \boldsymbol{w}) \times \boldsymbol{\mu}_k$
    $\boldsymbol{\sigma}^2_{k+1} = \boldsymbol{w} \times \bar{\boldsymbol{\sigma}}'^2 + (1 - \boldsymbol{w}) \times \boldsymbol{\sigma}^2_k$
  end for
  return optimized $\boldsymbol{\theta} = \boldsymbol{\mu}_{K+1}$
```

In order to determine which parameters should be updated to improve the performance for $\boldsymbol{s}^*$, without a loss of performance elsewhere, LCEPS computes a weight vector $\boldsymbol{w}$ such that $w_j \in [0,1]$ reflects the importance of the j^{th} policy parameter for the task associated to $\boldsymbol{s}^*$. The specific implementation of this weighting process for $\pi_{\boldsymbol{\theta}}$ can be found in Fig. 4. Then, $\boldsymbol{\theta}$ samples are drawn using the current normal distribution except

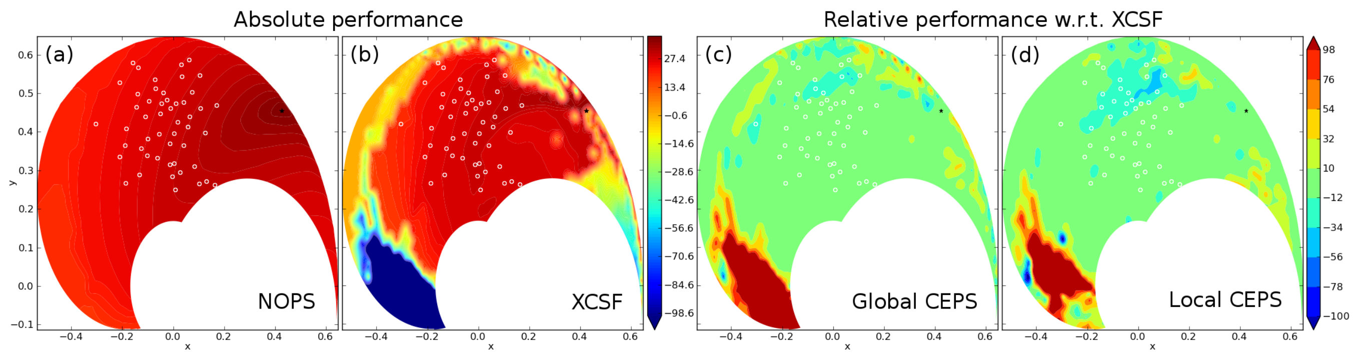

Fig. 3. Absolute performance of (a) the NOPS and (b) $\pi_{\boldsymbol{\theta}_0}$, and relative performance of $\pi_{\boldsymbol{\theta}}$ optimized by either (c) GCEPS and (d) CEPS with respect to (b) $\pi_{\boldsymbol{\theta}_0}$. The performance is represented given the target position, obtained by interpolating the performances for a large grid of targets. The learning target positions are indicated by white circles, and the starting position by a star. The performance is computed according to (1) and represented as a color according to the right-hand side scale.

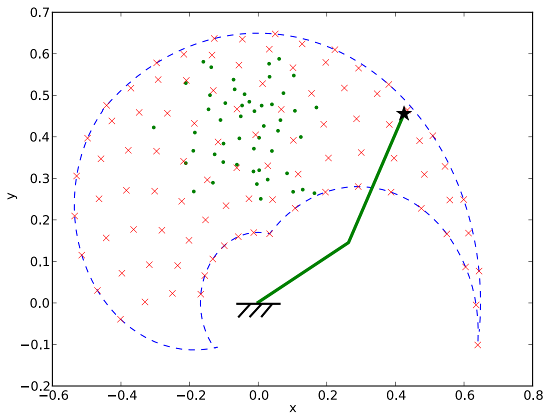

Fig. 5. The arm workspace. The reachable space is delimited by a spiral-shaped envelope. The two segments of the arm are represented by two bold lines. The initial configuration is the one represented. Learning targets are indicated by dots and testing targets by crosses.

for the parameters of null weight, for which the variance is set to 0. The corresponding $\pi_{\boldsymbol{\theta}}$ are evaluated by performing an episode and fed to the CEM. Finally, the result of this "local" CEM iteration is used to update the $\boldsymbol{\theta}$ distribution proportionally to the weight vector: the more relevant the parameter, the more it is taken into account in the new distribution.

III. Experimental design

We now illustrate the use of XCSF and CEPS for learning to control an arm to reach a given point 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. Experiments

We perform three experiments.

1) Generalization with XCSF: First, we create two sets of target positions, $\mathcal{L}$ for learning versus $\mathcal{T}$ for testing. The $\mathcal{L}$ set 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. The $\mathcal{T}$ set contains 94 targets in a 10×10 grid over the articular space (NOPS diverged for 6 of them, which were excluded). All trajectories start from the same point in the upper-right part of the reachable space (see Fig. 5). The NOPS is used to generate one trajectory for each target in $\mathcal{L}$. Then, XCSF learns from the actions of the NOPS for targets in $\mathcal{L}$, using the generated trajectories, and is tested as a policy on both sets. The condition and prediction spaces contain states $\boldsymbol{s}$. 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) Adaptation to a new target: Second, starting back from $\pi_{\boldsymbol{\theta}_0}$, we compare CEPS variants for improving the policy on a single target. This experience is performed for two targets: target A is located at the top-right of the workspace ($x = 0.2, y = 0.5$) and target B in the bottom-left corner ($x = -0.3, y = -0.5$).

3) Adaptation over the workspace: Third, we apply both CEPS variants for improving $\pi_{\boldsymbol{\theta}_0}$ over $\mathcal{T}$. For LCEPS, we select a different target from $\mathcal{T}$ after each iteration.

B. Experimental set-up and Parameters

We use the JavaXCSF [18] 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 230 minutes. Each CEM iteration contains $N = 100$ 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 $\sigma^2_{noise} = 10^{-3}$.

IV. Results

In this section, we first study whether the policy learned with XCSF is similar to the one obtained with the NOPS for

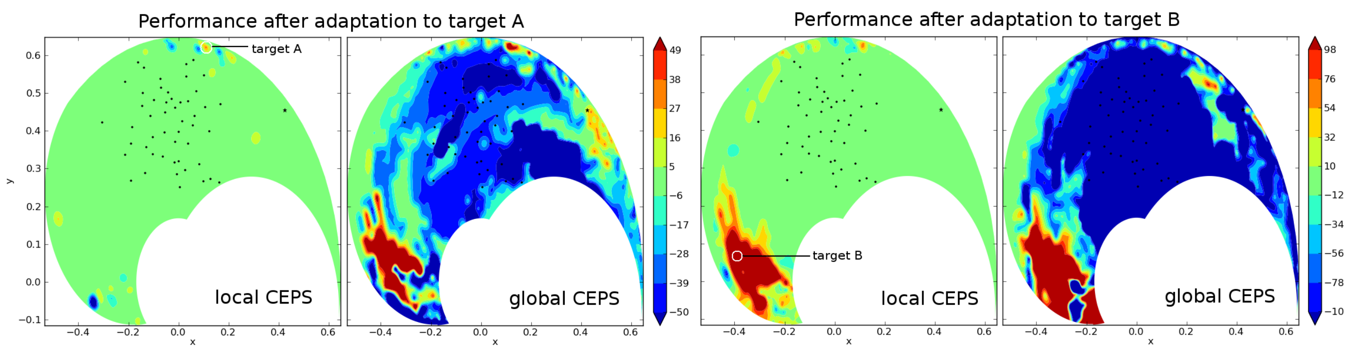

Fig. 6. Relative performance after applying GCEPS and LCEPS for adaptation of the XCSF policy to either target A or B.

TABLE I
MEAN AND STANDARD DEVIATION OF THE PERFORMANCE OVER TARGETS, BOTH FOR $\mathcal{L}$ AND $\mathcal{L}$

	$\mathcal{L}$	$\mathcal{T}$
NOPS	28.22 ± 2.06	27.46 ± 3.73
XCSF	27.45 ± 2.35	2.44 ± 46.03
GCEPS	22.96 ± 11.50	12.76 ± 22.28
LCEPS	25.19 ± 7.97	7.52 ± 38.76

TABLE II
MEAN AND STANDARD DEVIATION OF THE PERFORMANCE OVER TARGETS A AND B AND OVER $\mathcal{L}$

Algo.	target A	$\mathcal{L}$	target B	$\mathcal{L}$
NOPS	28.01	28.22 ± 2.06	21.17	28.22 ± 2.06
XCSF	-30.59	27.45 ± 2.35	-204.3	27.45 ± 2.35
GCEPS	27.98	-14.32 ± 12.49	22.88	-293.2 ± 169.7
LCEPS	26.14	27.40 ± 2.39	16.68	27.63 ± 2.01

the same targets, how well XCSF generalizes over different targets, and whether GCEPS is able to improve the performance of the learned policy. Then we compare the performance of LCEPS versus GCEPS, focusing on the improvement where generalization resulted in a poor performance and on the impact on the performance where it was already good.

A. *Performance of XCSF policy and generalization*

Results presented in this section are similar to those presented in [9], though we used only 500 classifiers instead of 6400 in [9]. The average running time to get one trajectory from the NOPS is $\sim$ 10 minutes. From $\pi_{\boldsymbol{\theta}_0}$, it is $\sim$ 30 milliseconds. Fig. 3(a) shows the performance of the NOPS as a function of the target position, obtained by interpolating the performance of the NOPS trajectories for all targets in $\mathcal{T}$. Fig. 3(b) shows the performance of one representative $\pi_{\boldsymbol{\theta}_0}$, which is very close to the NOPS performance for targets located near $\mathcal{L}$. One can see that the relative performance decreases with the distance to $\mathcal{L}$.

Fig. 3(c) shows the performance of $\pi_{\boldsymbol{\theta}}$ after applying GCEPS to optimize its parameters. One can see that the performance is globally improved over the whole workspace. 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. Results in Table I confirm quantitatively this visual feeling.

B. *Performance of the GCEPS versus LGCEPS*

The rationale behind proposing LCEPS was to provide a way to improve the performance locally, where it is needed, without disrupting the performance over targets for which the controller is already good. In order to evaluate this specific property, we performed the experiments described in Section III-A2. The results of these experiments can be visualized in Fig. 6, that gives the relative performance of the controller resulting from LCEPS with respect to $\pi_{\boldsymbol{\theta}_0}$. In Fig. 6(a), one can see that the performance on target A is increased without major changes elsewhere, despite some local decrease of performance in areas that are not close in the Cartesian workspace but in fact correspond to related articular configurations. Fig. 6(c) show the same for target B, with a much larger increase in performance around the target. This can be explained by the fact that $\pi_{\boldsymbol{\theta}_0}$ was particularly far from optimality around target B. The numerical results corresponding to this adaptation are shown in Table IV-B.

Interestingly, with GCEPS, one cannot train the controller over a specific target while measuring the performance over the whole workspace. If we evaluate the controller just on one target, then GCEPS optimizes the whole classifier population so as to maximize the performance on that specific target, which results in a very large disruption of performance over the workspace, as shown in Fig. 6(b) and (d).

Thus LCEPS is capable of improving the performance where needed, with only minor effects on the performance elsewhere. Furthermore, performing iterations of LCEPS over a set of targets is much faster than an iteration of GCEPS over the same set because LCEPS optimizes for a restricted set of parameters.

Given these positive results, we now compare the performance of GCEPS and LCEPS over $\mathcal{T}$. In order to do so, we apply LCEPS by choosing each target in $\mathcal{T}$ in sequence, as we do when performing a GCEPS iteration. Results can be visualized in Fig. 7 and extracted from a comparison between Fig. 3(c) and Fig. 3(d), as well as from Table I.

As one can see, LCEPS is not significantly better than GCEPS over the whole workspace, even if the good performance in the training region is more preserved. This is mainly because iterating over all targets as in GCEPS does not make full profit of the capabilities of LCEPS. In the future, instead of iterating over all targets with LCEPS, we plan to combine it with an *active learning* strategy that will spend more computational

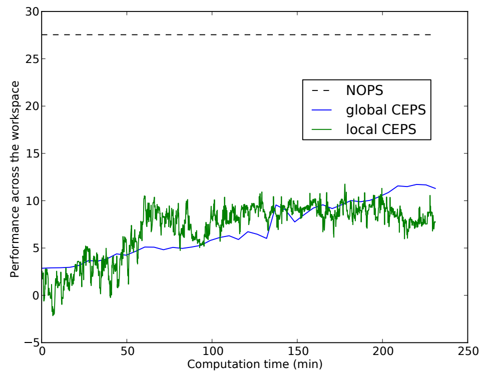

Fig. 7. Performance of the NOPS and $\pi_{\boldsymbol{\theta}}$ adapted by either GCEPS or LCEPS, as a function of time.

resources on targets where the performance can be more improved, and less resources where the performance is already good. Defining this *active learning* strategy is the matter of immediate future work.

V. Conclusion

In this paper, we have shown that learning and optimizing a parametric controller is a convincing alternative to using optimal control methods that are generally too expensive for real-time application in robotics. Here, using $\pi_{\boldsymbol{\theta}_0}$ was about 20000 times faster than using the NOPS. The resulting controller is fast and the generalization capability of LfD methods makes the learning process reasonably easy in practice. Furthermore, we have shown that using a stochastic optimization algorithm could further improve the policy, particularly in the areas where the performance is far from optimal.

However, from our empirical results, one can see that the obtained performance after a limited stochastic optimization period is still not close enough to the performance one gets with an OC method. There are two possible answers to this limitation. One consists in considering a *life-long learning* approach, where the system is optimizing its controller parameters throughout his life-time, yielding the slow convergence to a better optimum. Another option consists in learning a model of the plant, as presented in [15], [17] for instance, and use this model to improve the policy offline with virtual experiments based on the learned model. This will also help scaling the method to systems with more dimensions.

Our most immediate future work will consist in 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 with as few effort as possible. With the methods presented here, we will adapt in real-time the controller of the robot on-line to specific patients [10]. On a longer 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, D. E. Goldberg, and P.-L. Lanzi. Computational Complexity of the XCS Classifier System. *Foundations of Learning Classifier Systems*, 51:91–125, 2005.

[3] 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.

[4] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 8(1):28–46, 2004.

[5] V. Heidrich-Meisner and C. Igel. Similarities and differences between policy gradient methods and evolution strategies. In *Proceedings of the 16th European Symposium on Artificial Neural Networks (ESANN)*. Citeseer, 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. *Advances in Neural Information Processing Systems (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. of IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS-2010)*, 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. Reinforcement learning of motor skills with policy gradients. *Neural networks : the official journal of the International Neural Network Society*, 21(4):682–97, 2008.

[12] 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.

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

[14] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1(2):127–190, 1999.

[15] C. Salaün, V. Padois, and O. Sigaud. Control of redundant robots using learned models: an operational space control approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 878–885, 2009.

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

[17] P. Stalph, J. Rubinsztajn, O. Sigaud, and M. Butz. A Comparative Study: Function Approximation with LWPR and XCSF. In *Proceedings of the 13th International Workshop on Advances in Learning Classifier Systems*, 2010.

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

[19] 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.

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

[21] 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.

[22] S. W. Wilson. Classifiers that Approximate Functions. *Natural Computing*, 1(2-3):211–234, 2002.