Adaptive Exploration for Continual Reinforcement Learning

Freek Stulp
Cognitive Robotics, École Nationale Supérieure de Techniques Avancées (ENSTA-ParisTech), Paris, France
FLOWERS Research Team, INRIA Bordeaux Sud-Ouest, Talence, France
freek.stulp@ensta-paristech.fr

***Abstract*— Most experiments on policy search for robotics focus on isolated tasks, where the experiment is split into two distinct phases: 1) the learning phase, where the robot learns the task through exploration; 2) the exploitation phase, where exploration is turned off, and the robot demonstrates its performance on the task it has learned. In this paper, we present an algorithm that enables robots to continually and autonomously alternate between these phases. We do so by combining the 'Policy Improvement with Path Integrals' direct reinforcement learning algorithm with the covariance matrix adaptation rule from the 'Cross-Entropy Method' optimization algorithm. This integration is possible because both algorithms iteratively update parameters with probability-weighted averaging. A practical advantage of the novel algorithm, called PI^2-CMA, is that it alleviates the user from having to manually tune the degree of exploration. We evaluate PI^2-CMA's ability to continually and autonomously tune exploration on two tasks.**

I. Introduction

The exploration/exploitation trade-off in reinforcement learning refers to two opposing objectives: 1) explore the environment to determine a policy that minimizes costs; 2) exploit what has been learned. For instance, to learn that bumping into an obstacle leads to a penalty, a robot must first explore and make this experience. But once this is known, the robot should avoid this behavior [12].

In policy improvement for robotics, this trade-off is usually achieved by separating an experiment into two distinct phases: 1) learn the task through trial-and-error exploration; 2) stop learning by turning off exploration, and demonstrate the learned skill. As experiments usually focus on learning isolated tasks [6], [8], [11], this approach is feasible. But it is not suitable for life-long learning scenarios, as it limits the ability to adapt to changing tasks or environments.

In this paper, we focus on continual reinforcement learning (RL), and present the PI^2-CMA algorithm, which allows the robot to continually determine the exploration/exploitation trade-off over time. With PI^2-CMA, the robot automatically reduces the degree of exploration when the task is learned, allowing the robot to exploit what it has learned. On the other hand, the algorithm is also able to increase exploration, for instance to re-learn a task when it has changed.

We thank the three anonymous reviewers for their thorough reviews; in particular for providing several novel insights into the relation of our work to gradient-based methods and the Natural Evolution Strategy. This work is supported by the French ANR program (ANR 2010 BLAN 0216 01), more at `http://macsi.isir.upmc.fr`

PI^2-CMA achieves this by combining update rules from direct RL and evolutionary optimization strategies. The basis of the algorithm is the *Policy Improvement with Path Integrals* algorithm (PI^2), which enable robust, efficient learning in continuous, high-dimensional action spaces [11], but which has a constant degree of exploration. Adaptive exploration arises from integrating the covariance matrix adaptation rule from the *Cross-Entropy Method* (CEM) in PI^2. Hence PI^2-CMA's name, for *Policy Improvement with Path Integrals and Covariance Matrix Adaptation*. Even though CEM and PI^2 are derived from very different principles, combining these algorithms is feasible as they both iteratively update parameters with *probability-weighted averaging*.

In summary, the main advantages of PI^2-CMA are: 1) the degree of exploration is continuous, and there is no discrete transitions between the exploration and exploitation phase; 2) the robot determines the degree of exploration autonomously, based on the same trials used to optimize the policy parameters; 3) it alleviates the user from having to manually tune the degree of exploration, the only parameter in PI^2 which is not trivial to tune.

The rest of this paper is structured as follows. In the next section, we present related work, and describe the CEM and PI^2 algorithms. In Section III we integrate PI^2 and CEM into the PI^2-CMA algorithm. Section IV then evaluates PI^2-CMA on two separate tasks. We conclude with Section V.

II. Related Work

Reinforcement Learning problems are traditionally modelled as discrete Markov Decision Processes (MDPs), which have a discrete state and action space. Most research on adaptive exploration has been done in the context of discrete MDPs, which has lead to adaptive exploration algorithms such as E^3 [3], R-MAX [2], and others [12]. However, the *curse of dimensionality* and the discrete nature of MDPs makes it difficult to apply it to the high-dimensional, continuous spaces typically found in robotic control tasks.

An alternative to discrete state and action spaces is use a parameterized policy $\pi(\boldsymbol{\theta})$, and search directly in the space of the parameters $\boldsymbol{\theta}$ to find the optimal policy $\pi(\boldsymbol{\theta}^*)$. REINFORCE was an early direct reinforcement learning algorithm [13], and especially Natural Actor-Critic [6] demonstrated that this approach is applicable to robotics tasks. However, these algorithms are based on estimating a gradient from the trials, which cannot always be done robustly with a limited number of trials, noisy data, or discontinuous cost functions. Also, there are several algorithmic parameters which are difficult to tune by hand.

Instead of estimating a gradient, PI^2 and PoWER therefore use probability-weighted averaging [11], [4]. Interestingly enough, an almost identical covariance matrix update rule was derived for PoWER (in Appendix A.3 of [4]), but not included in the PoWER algorithm. In PoWER, the immediate costs must behave like an improper probability, i.e. sum to a constant number and always be positive. This can make the design of cost functions difficult in practice. PI^2 places no such constraint on the cost function, which may be discontinuous. When a cost function is compatible with both PoWER and PI^2, they perform essentially identical [11]. In a different context [5], also update the mean parameters $\boldsymbol{\theta}$ and exploration matrix $\sigma^2\mathbf{I}$ (Equation 36/37), but only update a scalar step-size parameter σ, rather than the full covariance matrix. The Fisher information matrix enables gradient-based algorithm to find a more direct path to the optimal solution in parameter space [6], and, although not investigated well from this perspective, may also be considered a form of adaptive exploration. The relationship between the natural gradient and probability-weighted averaging was recently made explicit through the framework of Information-Geometric Optimization [1].

An excellent discussion of the relationship between direct reinforcement learning algorithms (such as PI^2) and evolution strategies (such as CEM) is given by Rückstiess et al. [8], where extensive empirical comparisons between several gradient-based methods in both fields are made. The focus in our paper is on methods based on probability-weighted averaging (e.g. PoWER, PI^2) rather than gradients (e.g. REINFORCE, NAC), as these have proven to be superior in the context of direct RL for robotics problems [11]. Rückstiess et al. [8] motivate and empirically demonstrate the advantages of searching in parameter space for gradient methods, e.g. as in PGPE and NES. By using constant exploration noise over time, PI^2-CMA demonstrates the same for probability-weighted averaging. The superior performance of PGPE/NES and PI^2-CMA may thus have a common cause.

A. *Cross-Entropy Method (*CEM*)*

Given a n-dimensional parameter vector $\boldsymbol{\theta}$ and a cost function $J : \mathbb{R}^n \mapsto \mathbb{R}$, the Cross-Entropy Method (CEM) for optimization searches for the global minimum through iterative sampling from and updating of a probability distribution [7]. A commonly used distribution is a multi-variate Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}, \Sigma)$ with parameters $\boldsymbol{\theta}$ (mean) and Σ (covariance matrix). With this distribution, one iteration of CEM is implemented as: **1. sample** – Take K samples $\boldsymbol{\theta}_{k=1\dots K} \sim \mathcal{N}(\boldsymbol{\theta}, \Sigma)$ **2. evaluate** – Determine the cost $J(\boldsymbol{\theta}_k)$ of each sample. **3. sort** – Sort the samples in ascending order w.r.t. the costs $J(\boldsymbol{\theta}_k)$. **4. update** – Recompute the distribution parameters, based only on the first K_e 'elite' samples in the sorted list, as in (1) and (2). Throughout this paper, it will be useful to think of CEM as performing *probability-weighted averaging*, where the elite samples have probability $1/K_e$, and the non-elite have probability 0. One such iteration of CEM is visualized in Fig. 1, along with the assignment of probabilities to the elite samples.

$$\boldsymbol{\theta}^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} \boldsymbol{\theta}_k \tag{1}$$

$$\Sigma^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} (\boldsymbol{\theta}_k - \boldsymbol{\theta})(\boldsymbol{\theta}_k - \boldsymbol{\theta})^\intercal \tag{2}$$

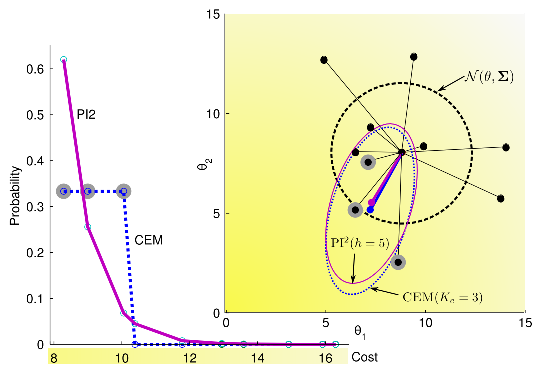

Fig. 1. Visualization of an update with CEM and PI^2. The upper right graph shows the 2D parameter space, with the current Gaussian distribution (dashed black), and $K = 10$ random samples taken from it. The cost of a sample is its distance to the origin in Cartesian space. The lower left graph shows the mapping from cost to probability for CEM and PI^2. For CEM, the elite samples are highlighted in gray.

B. *Policy Improvement with Path Integrals (PI^2)*

PI^2 is derived from first principles of optimal control, and gets its name from the application of the Feynman-Kac lemma to transform the Hamilton-Jacobi-Bellman equations into a so-called path integral, which can be approximated with Monte Carlo methods [11]. Rather than focussing on its derivation from first principles of stochastic optimal control, which is presented extensively in [11], we provide a post-hoc interpretation of the resulting update rule. The PI^2 algorithm is listed in Algorithm 1.

As in CEM, PI^2 takes K samples $\boldsymbol{\theta}_{k=1\dots K}$ from a Gaussian distribution. In PI^2, the vector $\boldsymbol{\theta}$ represents the parameters of a policy, which, when executed, yields a trajectory $\boldsymbol{\tau}_{i=1\dots N}$ with N time steps. This multi-dimensional trajectory may represent the joint angles of a n-DOF arm, or the 3-D position of an end-effector. So far, PI^2 has mainly been applied to policies represented as Dynamic Movement Primitives (DMPs) [11], where $\boldsymbol{\theta}$ determines the shape of the movement, i.e. the trajectory between the start (x_o) and end-point (the goal g) of the movement. When applied to DMPs, it is not necessary to take a different parameter sample $\boldsymbol{\theta}_{k,i}$ at each time step i, but it suffices to take one sample $\boldsymbol{\theta}_k$ before the execution [10].

Although the search space is in $\boldsymbol{\theta}$, the costs are defined in terms of the trajectory $\boldsymbol{\tau}$ generated by the DMP when it is integrated over time. The cost of a trajectory is determined by evaluating J for every time step i, where the cost-to-go of a trajectory at time step i is defined as the sum over all future costs $S(\boldsymbol{\tau}_{i,k}) = \phi_N + \sum_{j=i}^{N} q_{j,k}$.

Most policy improvement algorithms share the sampling and evaluation step above. However, the most crucial part is how the parameters $\boldsymbol{\theta}$ are updated. As in CEM, PI^2 also uses the principle of probability-weighted averaging to perform this update. For each time step i, PI^2 therefore computes the probability of a trajectory at i, by exponentiating the cost-to-go $S(\boldsymbol{\tau}_i)$, as in line 18 of Alg. 1. This assigns high probability to low-cost trials, and vice versa, as visualized in Fig. 1. The key step is then to compute the new policy parameters by performing *probability-weighted averaging* on the samples, as in line 21. Samples with low cost, and thus higher probability, therefore contribute more to the update than high cost samples. Using probability-weighted averaging avoids having to estimate a gradient, which can be difficult for noisy and discontinuous cost functions.

In line 21, a different parameter update $\boldsymbol{\theta}_i^{new}$ is computed for each time step i. To acquire the single parameter update $\boldsymbol{\theta}^{new}$, the final step is therefore to average over all time steps (line 26). This average is weighted such that earlier parameter updates in the trajectory contribute more than later updates, i.e. the weight at time step i is $T_i = (N-1)/\sum_{j=1}^{N}(N-1)$. The intuition is that earlier updates affect a larger time horizon and have more influence on the trajectory cost.

III. The PI^2-CMA algorithm

When comparing CEM and PI^2, there are several interesting similarities[1]. Both iteratively update the parameters of a distribution, both perform exploration by sampling parameters from a Gaussian distribution, and both use probability-weighted averaging to update the parameter distributions. It is striking that these algorithms, although having been derived from very different principles, have converged to almost identical parameter update rules.

There are also some important differences between CEM and PI^2. First of all, in policy improvement algorithms, $\boldsymbol{\theta}$ represents the parameters of a policy, which must be integrated over time to determine the costs. This temporal component requires us to compute a parameter update for each time step, and perform temporal averaging, which is not required in CEM.

Second, the mapping from costs to probabilities is different. CEM implements a cut-off value for 'eliteness': you are either elite ($P_k = 1/K_e$) or not ($P_k = 0$). PI^2 rather considers eliteness to be a continuous value that is inversely proportional to the cost of a trajectory, where the 'eliteness' parameter determines the slope of this inverse. Our empirical comparison of the CEM and PI^2 eliteness mappingdemonstrate that the PI^2 weighting scheme leads to slightly quicker convergence [10] . Fig. 1 visualizes an example where the different mappings lead to very similar distribution updates.

We now turn to the most interesting and relevant difference between the algorithms. In CEM, both the mean and covariance of the distribution are updated, whereas PI^2 only updates the mean. In this paper, we insert the covariance matrix adaptation rule from CEM in Eq. 2 into PI^2, thereby replacing the probabilities $1/K_e$ with the probabilities P_k as computed by PI^2. Thus, rather than having a fixed covariance matrix, PI^2 now adapts Σ based on the observed costs for the trials, as depicted in Fig. 1. This novel algorithm – 'Path Integral Policy Improvement with Covariance Matrix Adaptation' (PI^2-CMA) – is listed in Alg. 1.

```
input :
        $\boldsymbol{\theta}$ ;                                        initial parameter vector
1       $q_i$ and $\phi_N$ ;                           immediate and terminal cost function
2       $\lambda^{\text{init}}, \lambda^{\text{min}}, \lambda^{\text{max}}$, ;          exploration level (initial,min,max)
3       $K$ ;                                          number of roll-outs per update
4       $h$ ;                                          eliteness parameter

5  $\Sigma = \lambda^{\text{init}}\mathbf{I}$
6  while true do
7      Exploration: sample parameters and execute policies
8      foreach $k$ in $K$ do
9          $\boldsymbol{\theta}_k \sim \mathcal{N}(\boldsymbol{\theta}, \Sigma)$
10         $\boldsymbol{\tau}_k$ = executepolicy($\boldsymbol{\theta}_k$)
11     end
12     Compute parameter update for each time step
13     foreach $i$ in $N$ do
14         Evaluation: compute probability for each time step and trial
15         foreach $k$ in $K$ do
16             $S(\boldsymbol{\tau}_{i,k}) = \phi_{N,k} + \sum_{j=i}^{N} q_{j,k}$
17             $E(\boldsymbol{\tau}_{i,k}) = e^{\left(\frac{-h(S(\boldsymbol{\tau}_{i,k})-\min_l(S(\boldsymbol{\tau}_{i,l})))}{\max_l(S(\boldsymbol{\tau}_{i,l}))-\min_l(S(\boldsymbol{\tau}_{i,l}))}\right)}$
18             $P(\boldsymbol{\tau}_{i,k}) = \frac{E(\boldsymbol{\tau}_{i,k})}{\sum_{l=1}^{K} E(\boldsymbol{\tau}_{i,l})}$
19         end
20         Update: Probability-weighted averaging over $K$ trials
21         $\boldsymbol{\theta}_i^{new} = \sum_{k=1}^{K} [P(\boldsymbol{\tau}_{i,k})\ (\boldsymbol{\theta}_k - \boldsymbol{\theta})]$
22         $\Sigma_i^{new} = \sum_{k=1}^{K} [P(\boldsymbol{\tau}_{i,k})\ (\boldsymbol{\theta}_k - \boldsymbol{\theta})(\boldsymbol{\theta}_k - \boldsymbol{\theta})^\intercal]$
23         $\Sigma_i^{new}$ = boundcovar($\Sigma_i^{new}$)
24     end
25     Update: Temporal averaging over $N$ time steps
26     $\boldsymbol{\theta}^{new} = \frac{\sum_{i=0}^{N}(N-i)\boldsymbol{\theta}_i^{new}}{\sum_{l=0}^{N}(N-l)}$
27     $\Sigma^{new} = \frac{\sum_{i=0}^{N}(N-i)\Sigma_i^{new}}{\sum_{l=0}^{N}(N-l)}$
28 end
```

Algorithm 1: The PI^2 and PI^2-CMA update rule for a 1-D parameterized policy. The green lines (line 22, 23 and 27) only part of PI^2-CMA, not PI^2.

Because a covariance matrix update is computed for each time step i (line 22), we need to perform temporal averaging for Σ (line 27), just as we do for the mean $\boldsymbol{\theta}$ (line 26).

A. *Bounds on the covariance matrix*

In the original PI^2 algorithm, the covariance matrix Σ does not change during learning. It is initially set to $\Sigma = \lambda\mathbf{R}^{-1}$, where $\mathbf{R}$ is the control cost matrix[2]. The value of λ must

[1]For a more extensive comparison between PI^2 and CEM (as well as Covariance Matrix Adaptation - Evolutionary Strategy – CMA-ES [1]), we refer to [10]. This paper extends our previous work by: 1) demonstrating PI^2-CMA's capability to re-adapt to changing tasks; 2) evaluating PI^2-CMA on a more challenging, dynamic robotic task.

[2]By enabling covariance matrix updating, we violate the constraint $\Sigma = \lambda\mathbf{R}^{-1}$. Although the derivation of PI^2 is no longer possible with this violation, the resulting algorithm is not affected. It does not seem to have a negative effect in practice, as our results show. Also, it allows us to set $\mathbf{R} = 0$, and penalize accelerations directly through the immediate costs.

be tuned manually, and influences the convergence behavior of the algorithm, as our experiments in Section IV show. In PI^2-CMA, the initial Σ is set to $\Sigma = \lambda^{\text{init}}\mathbf{I}_B$, where B is the number of basis functions used in the DMP.

In PI^2-CMA, Σ is then subsequently adapted over time. A common problem with covariance matrix adaptation is premature convergence. To avoid degeneracy of Σ, we compute its eigenvalues, place a lower bound of λ^{min} on the eigenvalues, and reconstruct the bounded covariance matrix from the eigenvectors and the bounded eigenvalues. This procedure is implemented in the 'boundcovar' function in line 23 in Alg. 1. In Section IV-A.4, we demonstrate the effects of setting different values for λ^{min}. For robotics applications, it is also recommended to put an upper bound λ^{max} on the eigenvalues of Σ, as too much exploration might lead to dangerous behavior on the robot, e.g. reaching joint limits, or large accelerations.

IV. Experimental Evaluation

The rest of this paper is dedicated to two tasks, which are used to highlight several advantages of the PI^2-CMA algorithm. Section IV-A: a simple 1D viapoint task is used to illustrate the advantages of adaptive exploration.Section IV-B: demonstrate automatic re-learning on a more challenging task, involving a dynamically simulated humanoid robot which uses a bat to hit a ball in a specific target area.

A. An Illustratory Example

For this evaluation we use a via-point task, with a 1-dimensional DMP that has only two basis functions. The parameter search space is thus 2-dimensional. This simplicity has been chosen because having only a 2-D parameter space facilitates visualization on paper. Higher-dimensional problems are considered in the other experiments.

The goal of this task is for the 1-D output of the DMP to take the value 0.3 at time $0.5s$. The cost function (3) therefore penalizes the distance to the value $v = 0.3$ at $t = 0.5s$, as well as the acceleration at each time step. The acceleration penalty is divided by the number of time steps N to be independent of the movement duration. To simulate a change in the environment, we change v to 0.1 after 20 updates.

$$J_{t_i} = 10^3\delta(t_i - 0.5)|\ x_{t_i} - v\ | + \ddot{x}_{t_i}^2/N \tag{3}$$

The DMP is initialized as a 1s minimum-jerk trajectory. PI^2 performs $K = 15$ trials per update; a set of K trials is called an epoch. The eliteness parameter is also set to $h = 15$. For each learning session, we perform 40 updates, i.e. 600 trials. Each learning session is performed 10 times, and the values discussed in the following result sections always refer to the means over 10 learning sessions.

1) Results: Constant Exploration: Graphs C1-3 in Fig. 2 depict the results of PI^2 with constant exploration for $\lambda = \{20, 200, 2000\}$. The left graphs show how the parameters $\boldsymbol{\theta}$ and Σ change as learning progresses. The values at the axis ticks are only depicted in the bottom graphs for clarity, and are shared amongst all graphs in Fig. 2. The center of the distribution $\boldsymbol{\theta}$ moves from the initial point before learning to the center. When the viapoint changes, it moves even further towards the lower left corner of the graph, as annotated in C2. Since $\Sigma = \lambda\mathbf{I}$ is constant, the (circular) error ellipse does not change. We see that larger values of λ lead to larger updates of $\boldsymbol{\theta}$, which leads to quicker convergence.

This is confirmed by the learning curves to the right, where the dark blue graph shows the trajectory cost of the first trial of the 15 trials for each update, which is an evaluation trial executed without exploration noise. But fast convergence is not all we are interested in. Once converged, we also want to exploit what we have learned. To evaluate exploitation, we also plot the standard deviation over the 14 exploration trials for an update, as the light blue area around the learning curve ($\pm$ standard deviation). Higher degrees of exploration λ lead to larger differences in cost between the exploration trials, and thus a higher standard deviation. So although convergence is fast for $\lambda = 2000$, the variance of the costs in the exploration trials is very high. This means that the algorithm is not exploiting what it has learned, because it is still exploring a lot.

Summary: For constant exploration, there is an inherent trade-off between fast convergence, which is achieved with high exploration, and good exploitation of what has been learned, for which we need low exploration.

2) Results: Decaying Exploration: To combine the advantages of high and low degrees of exploration it is customary to decay exploration over time, where λ_u at update u is equal to $\lambda_u = \lambda^{\text{init}} \cdot D^{\text{\#updates}}$, with decay factor $0 < D < 1$. Graph D in Fig. 2 shows the results of this approach, with high initial exploration ($\lambda^{\text{init}} = 2000$), and a decay factor of $D = 0.8$. The left graph visualizes how the covariance matrix slowly becomes smaller over time. Apart from the learning curve, the right graph now also plots λ. The values at the axis ticks are again only in the bottom plot. Note that the plot is straight due to logarithmic y-axis for λ.

For the first 20 updates (before the via-point changes), we see quick convergence (due to high exploration at the beginning), as well as good exploitation once the task is learned (due to the low, decayed exploration). So do we now have the best of both worlds? Unfortunately not. The disadvantage is that when the task changes (e.g. changing the viapoint), exploration has decayed so much that the second task cannot be learned.

Summary: Decaying exploration works well when learning isolated task, but not for continual learning, where the environment or the task may change.

3) Results: Adaptive Exploration: Graph A1 shows the results of automatically updating Σ with PI^2-CMA, with λ^{init}=2000. As is the case for decaying exploration, convergence is quick in the beginning in graph A1, but when the task is learned and costs have converged, the degree of exploration λ becomes lower[3] and the variance in cost decreases. The main advantage in the context of continual

[3]Initially, Σ is diagonal ($\lambda^{\text{init}}\mathbf{I}$), but must no longer be so after the first update. For a given Σ_u, we therefore compute λ_u as its largest eigenvalue. This is the green graph plotted in graph A1-A4.

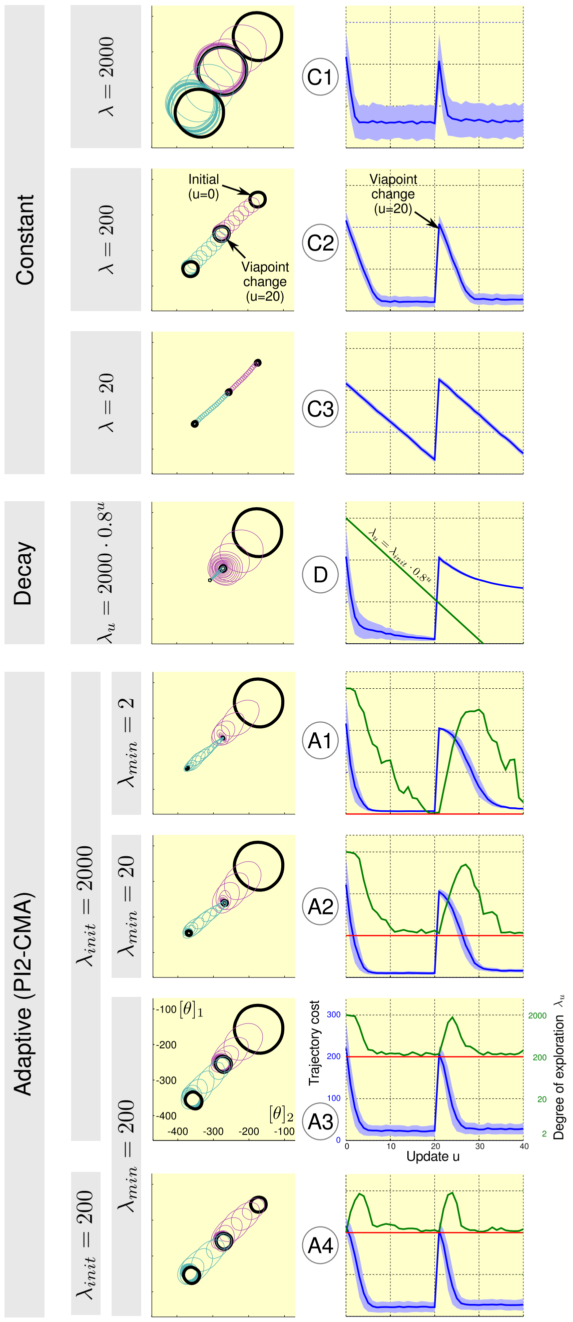

Fig. 2. Left: Policy distribution parameters at each update. Right: The learning curves (blue) and degree of exploration λ (green) at each update.

reinforcement learning is that when the task is changed after 20 updates, exploration automatically goes up again (green graph), and again decays when the new task is learned.

It is important to realize that we are not specifying explicitly that exploration should increase/decrease – it is a property that arises automatically from using probability-weighted averaging to update the covariance matrix[4]. If you are at a minimum, closer samples will have lower cost than samples that are further away, which leaves $\boldsymbol{\theta}$ approximately where it is, but causes the covariance matrix Σ to shrink.

[4]An alternative (heuristic) approach could be: increase exploration when the cost is low, decrease exploration when the cost is high. However this requires knowledge about what 'low' and 'high' costs are, for every task and in potentially changing environment. The exploration magnitude is, in contrast, independent of task and environment.

Summary: PI^2-CMA is able to switch autonomously between phases in which it learns and phases in which it predominantly exploits what it has learned.

4) Results: Influence of minimum degree of exploration λ^{min}: Graphs A1-3 all have λ^{init}=2000, but vary in the minimum degree of exploration $\lambda^{\text{min}} = \{2, 20, 200\}$, as described in Section III-A. Setting a low λ^{min} (A1) leads to less variance in cost once the costs have converged, which allows adaptive exploration to exploit more once the task is solved. In A2-3, where λ^{min} is higher, the variance in the exploration trials is also higher, even after having converged ($10 < u < 20$ for the first viapoint, and $35 < u < 40$ for the second). Setting a high λ^{min} leads to a faster increase in exploration if the task changes, which may lead to faster convergence after a task has changed. This effect can also be seen in graphs A1-3, after the viapoint change. For $\lambda^{\text{min}} = \{2, 20, 200\}$, the costs have approximately converged at update 38, 32, 27 respectively.

Summary: λ^{min} provides a trade-off between more exploitation after convergence (for lower λ^{min}) and faster re-adaptation when tasks change (for higher λ^{min}).

5) Results: Automatic Tuning of the Exploration Magnitude: Graphs A3 and A4 have the same minimum of $\lambda^{\text{init}} = 200$, but have a different initial exploration magnitude, i.e. $\lambda^{\text{init}} = \{200, 2000\}$. The main conclusion from these two graphs are that for $\lambda^{\text{init}} = 200$, the exploration quickly goes up until, after 5 updates, the learning curves and exploration magnitude develop almost identically. We have demonstrated this effect on a different 10-D task as well, and were able to show that even if the initial exploration magnitude is varied by 4 orders of magnitude, it still adapts and converges towards the same value [10]. PI^2-CMA is thus robust towards the value of λ^{init}, which means this parameter does not have to be tuned by the user.

B. Re-Adaptation to Changing Tasks

In this experiment, we show PI^2-CMA's re-adaptation capability in a more challenging dynamic task.

1) Evaluation Task: Inspired by [6], the goal in this task is for the robot to use a bat to hit a ball such that lands in a designated area. We use the SL simulation environment [9] to accurately simulate the CBi humanoid robot, as visualized in Fig. 3. We keep the torso of the robot fixed, and use only the 7 degrees of freedom of the right arm. The bat is fixed to the end-effector. The cost function for this task is:

$$J_{t_i} = 0.01 \sum_{d=1}^{7} \ddot{a}_{t_i}^2 / N + \phi \tag{4}$$

$$\phi = \begin{cases} \text{if in target area} & 0 \\ \text{else} & \text{distance to target area in m} \end{cases} \tag{5}$$

Where we penalize the acceleration of d^{th} joint $a_{t_i}^d$ to avoid high acceleration movement). We divide by the number of time steps N so as to be independent of the movement duration. The target area lies between -1 and -1.5m from the robot in the y direction, as visualized in Fig. 3.

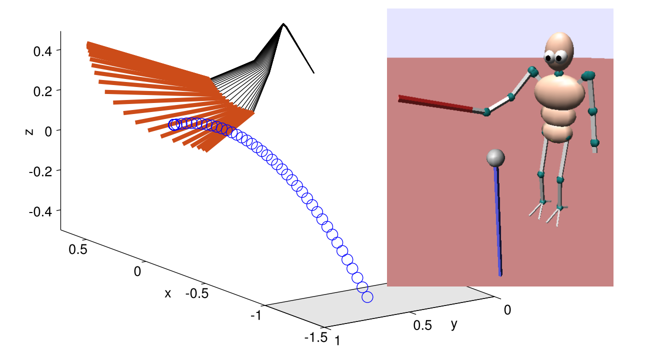

Fig. 3. The T-ball task for the CBi robot. The bat and ball trajectory are those as learned after 20 updates.

The DMP has 7 dimensions to control the 7 joint angles of the arm. Each dimension has 3 basis functions, and is initialized as a minimum-jerk trajectory of duration 1s from the start to the end pose as visualized in Fig. 3. The PI^2 parameters are $K = 10$, $h = 10$. Initially $\Sigma_{init} = \lambda^{\text{init}}\mathbf{I}_3$ with λ^{init}=20, and λ^{min}=0.02.

2) Results and Discussion: The learning curve and degree of exploration λ are depicted at the center of Fig. 4. The trajectory of the ball after 1, 20, 21, 40 updates is depicted in the top graph, and the trajectories of the end-point of the bat for the 10 exploration trials after 1, 20, 27 and 40 updates are depicted at the bottom.

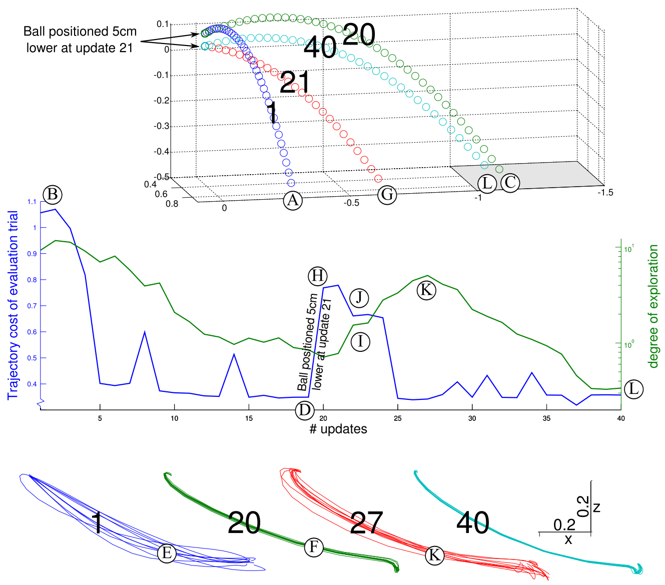

Fig. 4. The center graph depicts the cost of the evaluation trial and the degree of exploration as learning progresses. The trajectories of the ball (top plot) and the trajectory of the end-point of the bat (bottom plot) after 1, 20, 21/27 and 40 updates.

Initially, the ball lands far from the target area Ⓐ, which leads to high costs Ⓑ. After 20 updates, the ball lands in the target area Ⓒ (it does so for the first time after only 5 updates), and the costs are much lower Ⓓ, *and* exploration has been dramatically reduced Ⓓ (note logarithmic y-axis for λ). The lower exploration also becomes clear in the bottom plots, where the variance in the bat's movement is initially much higher Ⓔ than after 20 updates Ⓕ.

After update 20, we position the ball 5cm lower, which has several consequences: the ball no longer lands in the target area Ⓖ so costs immediately go up Ⓗ, after which exploration increases again Ⓘ and costs go down Ⓙ. After 27 updates, exploration reaches a maximum Ⓚ, and decreases again. After 40 updates, the costs and exploration are both very low Ⓛ. Note that because the penalty due to accelerations is quite high in this task (we recommend this for ballistic movements as required for T-ball), the costs do not converge as close to 0 as in the other task.

Summary: PI^2-CMA is able to switch autonomously between phases in which it learns and phases in which it predominantly exploits what it has learned.

V. Conclusion

Determining an appropriate degree of exploration for a good exploration/exploitation trade-off has extensively been studied in the context of reinforcement learning with discrete Markov decision processes [2], [3], [12]. In this paper, we combine the PI^2 and CEM algorithms to derive a *direct* reinforcement learning algorithm, which explores directly in policy parameter space, and adapts the degree of exploration autonomously. As our experiments show, this alleviates the user from manually tuning this parameter, and allows the robot to both reduce exploration to increase exploitation *and* increase exploration to adapt to changing tasks.

Our future work aims at applying PI^2-CMA to dynamic tasks on physical robots. Given the ability of PI^2 to learn complex tasks on real robots, we are confident that PI^2-CMA can also successfully be applied to real robots.

References

[1] L. Arnold, A. Auger, N. Hansen, and Y. Ollivier. Information-geometric optimization algorithms: A unifying picture via invariance principles. Technical report, INRIA Saclay, 2011.

[2] R. Brafman and M. Tennenholtz. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, March 2003.

[3] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Mach. Learn.*, 49(2-3):209–232, 2002.

[4] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Machine Learning*, 84:171–203, 2011.

[5] J. Peters and S. Schaal. Learning to control in operational space. *I. J. Robotic Res.*, 27(2):197–212, 2008.

[6] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[7] R.Y. Rubinstein and D.P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, 2004.

[8] T. Rückstiess, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber. Exploring parameter space in reinforcement learning. *Paladyn. Journal of Behavioral Robotics*, 1:14–24, 2010.

[9] S. Schaal. The SL simulation and real-time control software package. Technical report, University of Southern California, 2007.

[10] F. Stulp and O. Sigaud, Path integral policy improvement with covariance matrix adaptation In *Proceedings of ICML*, 2012.

[11] E. Theodorou, J. Buchli, and S. Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, 11:3137–3181, 2010.

[12] S B. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie-Mellon University, 1992.

[13] R. Williams. Simple statistical gradient-following algorithms for connectionist RL. *Machine Learning*, 8:229–256, 1992.