0

Adaptive Exploration through Covariance Matrix Adaptation Enables Developmental Motor Learning

Freek Stulp[1,2*],
Pierre-Yves Oudeyer[1,2†]

1 Robotics and Computer Vision
ENSTA-ParisTech
Paris, France

2 FLOWERS Team
INRIA Bordeaux Sud-Ouest
Talence, France

Abstract
The "Policy Improvement with Path Integrals" (PI^2) [25] and "Covariance Matrix Adaptation – Evolutionary Strategy" [8] are considered to be state-of-the-art in direct reinforcement learning and stochastic optimization respectively. We have recently shown that incorporating covariance matrix adaptation into PI^2– which yields the PI^2_{CMA} algorithm – enables adaptive exploration by continually and autonomously reconsidering the exploration/exploitation trade-off. In this article, we provide an overview of our recent work on covariance matrix adaptation for direct reinforcement learning [22–24], highlight its relevance to developmental robotics, and conduct further experiments to analyze the results. We investigate two complementary phenomena from developmental robotics. First, we demonstrate PI^2_{CMA}'s ability to adapt to slowly or abruptly changing tasks due to its continual and adaptive exploration. This is an important component of life-long skill learning in dynamic environments. Second, we show on a reaching task how PI^2_{CMA} subsequently releases degrees of freedom from proximal to more distal limbs as learning progresses. A similar effect is observed in human development, where it is known as 'proximodistal maturation'.

Keywords
reinforcement learning · covariance matrix adaptation · developmental robotics · adaptive exploration · proximodistal maturation

1. Introduction

Due to its generality, temporal abstraction, and ability to learn without models, reinforcement learning (RL) has been proposed as an appealing paradigm for organizing learning and behavior in developmental robotics [21]. Using RL in the context of robotics – and developmental robotics in particular – introduces several challenges, including scaling to high-dimensional continuous action spaces, being able to adapt to changing tasks and environments, and continually reconsidering the exploration/exploitation trade-off.

We have recently [24] proposed to address these challenges by incorporating covariance matrix adaptation, as used in for instance "Covariance Matrix Adaptation – Evolutionary Strategy" [8], into the state-of-the-art direct reinforcement learning algorithm "Policy Improvement with Path Integrals" (PI^2) [25]. PI^2_{CMA} inherits its robust, efficient learning in high-dimensional action spaces from PI^2, whereas its novel covariance matrix adaptation adapts exploration such as to achieve a good exploration/exploitation trade-off over time.

In this article, we provide an overview of this recent work [22–24], highlight its relevance to developmental robotics, and conduct further experiments to analyze the results. After discussing

*E-mail: freek.stulp@ensta-paristech.fr
†E-mail: pierre-yves.oudeyer@inria.fr

related work in Section 2, we present the PI^2_{CMA} algorithm – introduced in [24] – in Section 3. We then focus on using PI^2_{CMA} in the context of developmental robotics by investigating two complementary phenomena.

First, we demonstrate PI^2_{CMA}'s ability to automatically adapt to changing tasks [22] in Section 4. PI^2_{CMA} does so by determining the appropriate exploration magnitude *autonomously* – exploration decreases once a task has been learned (exploitation), but increases again automatically if the task or environment changes such that the task must be re-learned. This exploration behavior is not explicitly encoded in the algorithm, but is rather an emergent feature of updating the covariance matrix, which governs exploration, with probability-weighted averaging.

Second, Section 5 shows how PI^2_{CMA} spontaneously self-organizes a maturational structure while exploring the degrees-of-freedom of the motor space [23]. The algorithm automatically releases degrees of freedom from proximal to more distal limbs as learning progresses. A similar process is observed when infants learn to reach [3, 11]. PI^2_{CMA} achieves this smoothly and entirely through self-organization, rather than using discrete stages or pre-defining their order, as in [2, 4, 5].

2. Related Work

In this article, we demonstrate that covariance matrix adaptation enables adaptive exploration in a reinforcement learning context. Most research on adaptive exploration for reinforcement learning has been done in the context of discrete Markov Decision Processes (MDPs), which has lead to adaptive exploration algorithms such as E^3 [10], R-MAX [6], and others [26]. 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 [27], and especially Natural Actor-Critic [14] demonstrated that this approach is applicable to robotics tasks. The Fisher information matrix enables NAC to find a more direct path to the optimal solution in parameter space [14], and, although not investigated well from this perspective, may also be considered a form of adaptive exploration. In general, gradient-based algorithms must estimate 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. The relationship between the natural gradient and probability-weighted averaging, as used in PI^2 and PI^2_{CMA}, was recently made explicit through the framework of Information-Geometric Optimization [1].

Because our focus is not on the algorithms that are the foundation of PI^2_{CMA}, we have not been able to do justice to the sound derivations on which these algorithms are based. For a more in-depth discussion of covariance matrix adaptation – as in the CMA-ES– we refer to [8]. 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. For the full derivation, we refer to [25]. Related algorithms include Policy Gradients with Parameter-Based Exploration (PGPE) [20], the first direct policy search method with parameter exploration. Covariance matrix adaptation is applied to PGPE by Miyamae et al. [12] and also used in Natural Evolution Strategies [7]. Because these algorithms use a scalar reward/cost function, they are evolution strategies, and do not use a temporal averaging step as PI^2_{CMA}. An excellent discussion of the relationship between direct reinforcement learning algorithms and evolution strategies is given by Rückstiess et al. [16], where extensive empirical comparisons between several methods in both fields are made.

Several previous works have also considered mechanisms for progressive release of motor degrees of freedom, the focus of the experiment in Section 5. Some models have studied the impact of pre-defined and fixed stages of freeing and freezing DOFs [4]. Others have shown how the pace of the sequencing of discrete stages [5] or of the continuous increase of explored values of DOFs along a proximodistal scheme [2] could be adaptively and non-linearly controlled by learning progress and lead to efficient motor learning in high-dimensional robots. In this article, we have shown that without an explicit mechanism for motor maturation, such efficient maturational schedules, alternating freezing and freeing of DOFs, can be generated by PI^2_{CMA} entirely automatically.

In this respect, the work Schlesinger et al. [19] is most similar to ours, in that it also uses a kinematically simulated arm, and *"several constraints appear to 'fall out' as a consequence of a relatively simple trial-and-error learning algorithm."* [19], one of them being the locking of joints. Policies are represented as four-layer feedforward neural networks, which are trained through evolutionary learning. The main differences to our work is that we consider higher-dimensional systems – 10 DOF instead of 3DOF – and use one learning agent instead of a population of 100. On the other hand, we take only proprioceptive information into account (the current joint angles), whereas Schlesinger et al. [19] also consider visual feedback (an abstract image with 1×8 pixels) and tactile information (a boolean indicating contact with the object). The fact that such different policy representations and learning algorithms lead to similar emergent properties indicates that some deeper relationship between optimization and proximodistal maturation cause this emergence. A first step towards finding this relationship is presented in Section 5.1.

3. PI^2_{CMA} Algorithm

The PI^2_{CMA}, or "Policy Improvement with Path Integrals and Covariance Matrix Adaptation", was recently proposed in [24]. It is a combination of the PI^2 direct reinforcement learning algorithm [25] with covariance matrix adaptation, as used in for instance CMA-ES [8]. For a discussion of the similarities and differences between these algorithms we refer to [24].

The goal of PI^2_{CMA} is to optimize a set of policy parameters $\boldsymbol{\theta}$ with respect to a cost function $R(\boldsymbol{\tau}) = \phi_{t_N} + \sum_{t_i}^{t_N} r_t$, where $\boldsymbol{\tau}$ is a trajectory that starts at time t_0 in state $\mathbf{x}_{t_0}$ and ends at t_N. ϕ_{t_N} is the terminal reward at t_N, and r_t is the immediate reward at time t. For example, ϕ_{t_N} may penalize the distance to a goal at the *end* of a movement, and r_t may penalize the acceleration at each time step *during* a movement. To optimize $R(\boldsymbol{\tau})$, PI^2_{CMA} uses an iterative approach of exploring and updating in policy parameter space, as listed in Algorithm 1, which we now describe in more detail.

Exploration

PI^2_{CMA} first takes K samples from a Gaussian distribution $\boldsymbol{\theta}_{k=1\dots K} \sim \mathcal{N}(\theta_\mu, \boldsymbol{\Sigma})$ (line 1). The vector $\boldsymbol{\theta}$ represents the parameters of a policy, which for instance controls the sequence of desired joint angle of an arm, or desired x-coordinate of an end-effector. As the PI^2 algorithm on which it is based, PI^2_{CMA} therefore explores in policy parameter space, a concept first proposed by Sehnke et al. [20]. Executing the policy with parameters $\boldsymbol{\theta}_k$ yields a trajectory $\boldsymbol{\tau}$ with N time steps (line 1). An entire trajectory is referred to as $\boldsymbol{\tau}$, whereas $\boldsymbol{\tau}_i$ refers to the subtrajectory of $\boldsymbol{\tau}$, starting at t_i, and ending at t_N. In this nomenclature, $\boldsymbol{\tau}$ is therefore just a convenient abbreviation for $\boldsymbol{\tau}_0$. From now on, indices i and j refer to time steps, and k and l refer to trials. A *trial* or *roll-out* is the full trajectory resulting from executing the policy with parameters $\boldsymbol{\theta}_k$.

Parameter Update per Time Step: Probability-weighted Averaging

After exploration, a new parameter vector $\boldsymbol{\theta}^{\text{new}}_\mu$ is computed, which is expected to lead to a lower trajectory cost than the current θ_μ. This parameter update is done in two phases: 1) compute different parameters $\boldsymbol{\theta}^{\text{new}}_{\mu,i}$ for each of the N time steps i, using probability-weighted averaging; 2) compile the N updates $\boldsymbol{\theta}^{\text{new}}_{\mu,i}$ into one vector $\boldsymbol{\theta}^{\text{new}}_\mu$, using temporal averaging.

Although exploration is performed in the space of $\boldsymbol{\theta}$, costs are incurred by actually executing the policy, and are thus defined in

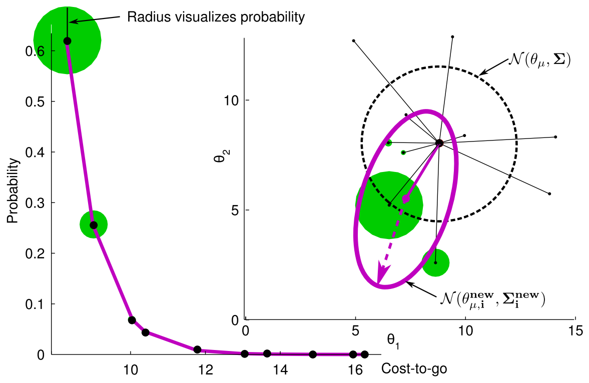

Figure 1. Visualization of a parameter update at one time step i. The upper right graph shows the 2D parameter space, with the current Gaussian distribution (dashed black), and $K = 10$ random samples taken from it. The lower left graph shows the mapping from cost to probability. In this illustratory example, the minimum $\boldsymbol{\theta}^*$ lies at [0,0]; samples $\boldsymbol{\theta}_k$ closer to [0,0] thus have lower costs, and therefore higher probabilities (represented by green circles). The new distribution $\boldsymbol{\theta}^{\text{new}}_{\mu,i}, \boldsymbol{\Sigma}^{\text{new}}{}_i$ for time step i (purple ellipse) is acquired through probability-weighted averaging.

terms of the trajectory $\boldsymbol{\tau}$ that results from this execution. PI^2_{CMA} optimizes the parameters $\boldsymbol{\theta}$ not only for the entire trajectory, but also for all subtrajectories $\boldsymbol{\tau}_{i=1\dots N}$ of $\boldsymbol{\tau}$. The cost of a subtrajectory $\boldsymbol{\tau}_i$ is computed as the sum over the costs throughout the rest of the trajectory starting at time step i: $S(\boldsymbol{\tau}_i) = \phi t_N + \sum_i^N r_{t_i}$. This is known as the *cost-to-go*, because it represents the accumulated 'cost to go' from i to the end of the trajectory. The underlying principle (the Bellman principle) is that a subtrajectory $\boldsymbol{\tau}_i$ starting at t_i can only be optimal if the subtrajectory $\boldsymbol{\tau}_{i+1}$ is optimal too.

The probability of each trajectory $P(\boldsymbol{\tau}_{i,k})$ is then computed by exponentiating the cost-to-go $S(\boldsymbol{\tau}_{i,k})$ of that trajectory at each time step (line 1). For illustration purposes, this transformation from cost to probability is depicted in Figure 1. Here, we see the $K = 10$ samples in a two-dimensional $\boldsymbol{\theta}$ space. The mapping from cost to probability is visualized in the lower-left graph. High-cost samples are assigned a low probability, and low-cost samples a high probability. This mapping follows directly from the PI^2 derivation, and may be interpreted as preferring trajectories with lower cost to occur with a higher probability. The parameter h determines the exact shape of the mapping from cost to probability.

The two core steps in PI^2_{CMA} (line 1 and line 1) are then to update the mean and covariance matrix of the sampling distribution by using *probability-weighted averaging*: $\boldsymbol{\theta}^{\text{new}}_\mu = \sum P_k \boldsymbol{\theta}_k$. Since low-cost samples have a higher probability, this means they

```
input :
    $\boldsymbol{\theta}^{\text{init}}$ ;                          initial parameter vector
    $J(\boldsymbol{\tau})$ ;                             cost function
    $\lambda^{\text{init}}, \lambda^{\text{min}}, \lambda^{\text{max}},$ ;      exploration level (initial,min,max)
    $K$ ;                                number of roll-outs per update
    $h$ ;                                eliteness parameter

$\theta_\mu = \boldsymbol{\theta}^{\text{init}}$
$\boldsymbol{\Sigma} = \lambda^{\text{init}}\mathbf{I}$
while true do
    Exploration: sample parameters and execute policies
    foreach $k$ in $K$ do
        $\boldsymbol{\theta}_k \sim \mathcal{N}(\theta_\mu, \boldsymbol{\Sigma})$
        $\boldsymbol{\tau}_k = \text{executepolicy}(\boldsymbol{\theta}_k)$
    end
    Compute parameter update for each time step
    foreach $i$ in $N$ do
        Evaluation: compute probability for each time step and trial
        foreach $k$ in $K$ do
            $S(\boldsymbol{\tau}_{i,k}) = \sum_{j=i}^{N} J(\boldsymbol{\tau}_{j,k})$
            $E(\boldsymbol{\tau}_{i,k}) = e^{\left(\frac{-h(S(\boldsymbol{\tau}_{i,k})-\min(S(\boldsymbol{\tau}_{i,k})))}{\max(S(\boldsymbol{\tau}_{i,k}))-\min(S(\boldsymbol{\tau}_{i,k}))}\right)}$
            $P\left(\boldsymbol{\tau}_{i,k}\right) = \frac{E(\boldsymbol{\tau}_{i,k})}{\sum_{l=1}^{K} E(\boldsymbol{\tau}_{i,l})}$
        end
        Update: Probability-weighted averaging over $K$ trials
        $\boldsymbol{\theta}_{\mu,i}^{\text{new}} = \sum_{k=1}^{K} \left[P\left(\boldsymbol{\tau}_{i,k}\right)\ \mathbf{M}_{i,k}(\boldsymbol{\theta}_k - \theta_\mu)\right]$
        $\boldsymbol{\Sigma}_i^{new} = \sum_{k=1}^{K} \left[P\left(\boldsymbol{\tau}_{i,k}\right)\ (\boldsymbol{\theta}_k - \theta_\mu)(\boldsymbol{\theta}_k - \theta_\mu)^{\intercal}\right]$
        $\boldsymbol{\Sigma}_i^{new} = \text{boundcovar}(\boldsymbol{\Sigma}_i^{new}, \lambda^{\text{min}}, \lambda^{\text{max}})$
    end
    Update: Temporal averaging over $N$ time steps
    $\boldsymbol{\theta}_{\boldsymbol{\mu}}^{\text{new}} = \frac{\sum_{i=0}^{N}(N-i)\boldsymbol{\theta}_{\mu,i}^{\text{new}}}{\sum_{l=0}^{N}(N-i)}$
    $\boldsymbol{\Sigma}^{new} = \frac{\sum_{i=0}^{N}(N-i)\boldsymbol{\Sigma}_i^{new}}{\sum_{l=0}^{N}(N-i)}$
end
```

Algorithm 1: The PI^2_{CMA} algorithm.

will contribute more to the update[1]. The resulting update is visualized in Figure 1. As we see, the distribution mean $\boldsymbol{\theta}_{\boldsymbol{\mu}}^{\text{new}}$ is now closer to the minimum, and the covariance matrix is also 'pointing' more towards to the minimum. Using probability-weighted averaging avoids having to estimate a gradient, which can be difficult for noisy and discontinuous cost functions.

Parameter Update: Temporal Averaging

In line 1, a different parameter update $\boldsymbol{\theta}_{\mu,i}^{\text{new}}$ is computed for each time step i. If the trajectory has 500 time steps, we therefore perform probability-weighted averaging 500 times. To acquire a single new parameter vector $\boldsymbol{\theta}_{\boldsymbol{\mu}}^{\text{new}}$, the final step is therefore to average over all time steps (line 1). 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 [25].

Covariance Matrix Updating: Lower Bounds

In PI^2_{CMA}, the initial covariance matrix $\boldsymbol{\Sigma}$ of the Gaussian distribution from which samples are taken is set to $\boldsymbol{\Sigma} = \lambda^{\text{init}}\mathbf{I}_B$. Here, B is the dimensionality of the policy parameter vector $\boldsymbol{\theta}$, which corresponds to the number of basis functions (cf. Section 4.2). In PI^2_{CMA}, $\boldsymbol{\Sigma}$ is then subsequently updated and adapted over time; one such update is illustrated in Figure 1. A common problem with covariance matrix adaptation is premature convergence *"bringing search, with respect to the short principal axes of* $\boldsymbol{\Sigma}$*, to a premature end."* [8]. Therefore, *"[i]n the practical application, a minimal variance [...] should be ensured"* [8]. To avoid such degeneracy of $\boldsymbol{\Sigma}$, 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[2]. This procedure is implemented in the 'boundcovar' function in line 1 in Alg. 1. In our experience, covariance matrix bounding is essential, as the algorithm usually prematurely converges without it[3].

From now on, we will refer to the 'exploration magnitude' as the largest eigenvalue λ of the covariance matrix $\boldsymbol{\Sigma}$. The length of the dashed arrow in Figure 1 represents the larges eigenvalue λ; the direction of the arrow is the eigenvector. Note that the initial covariance matrix $\boldsymbol{\Sigma} = \lambda^{\text{init}}\mathbf{I}_B$ has a 'largest' (they are all the same) eigenvalue of λ^{init}.

Multi-dimensional policies

Algorithm 1 is applied to the parameters of a 1-D policy. Optimizing the parameters of an M-dimensional policy, e.g. 7-D for the 7 joints of an arm, or 3-D for the end-effector position, is done by running the algorithm in parallel for each of the dimensions of the policy, with the same costs but different parameter vectors $\boldsymbol{\theta}_{m=1\dots M}$ and covariance matrices $\boldsymbol{\Sigma}_{m=1\dots M}$

4. Re-Adaptation to Changing Tasks

In this section, we first show how PI^2_{CMA} is able to adapt to changing tasks; an important component of life-long skill learning in dynamic environments. In Section 5, we then show how

[1] $\mathbf{M}_{t_j,k}$ *is a projection matrix onto the range space of* $\mathbf{g}_{t_j}$ *[25], which are the basis function activations, cf. (6)*

[2] *For robotics applications, we also recommend putting an upper bound* λ^{max} *on the eigenvalues of* $\boldsymbol{\Sigma}$*, as too much exploration might lead to dangerous behavior on the robot, e.g. reaching joint limits, too high accelerations. For the simulated experiments described in this article,* λ^{max} *was not used.*

[3] *More robust convergence may also be achieved by updating only the diagonal of the covariance matrix [15].*

PI^2_{CMA} automatically freezes and frees joints in an arm, thereby sequentially freeing joints in a proximodistal order.

4.1. Experiment 1: T-Ball

Inspired by [14], 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 [18] to accurately simulate the CBi humanoid robot, as visualized in Figure 2. 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.

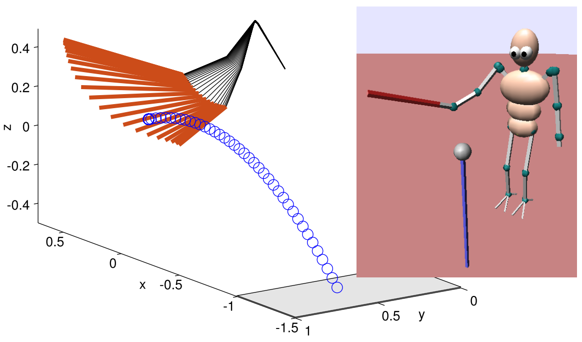

Figure 2. The T-ball task for the CBi robot. The bat and ball trajectory are those as learned after 20 updates.

The robot uses PI^2_{CMA} to optimize the shape parameters of a Dynamic Movement Primitive (DMP) [9]. The parameter of the DMP, the cost function of the task, and the PI^2_{CMA} parameters are summarized in Appendix A.

The learning curve and exploration magnitude λ are depicted at the center of Figure 3. 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.

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

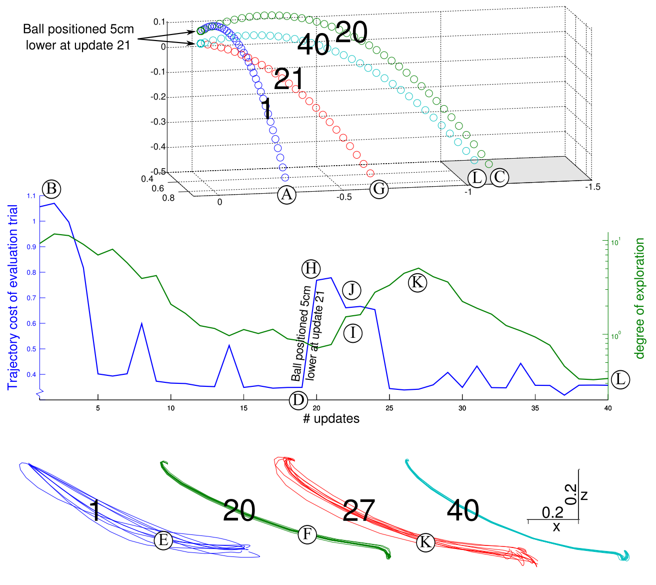

Figure 3. The center graph depicts the cost of the evaluation trial and the total exploration magnitude as learning progresses. The trajectories of the ball (top plot) and the trajectory of the end-point of the bat (bottom plot) after 1, 20, 21/27 and 40 updates.

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.

Conclusion

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. This is an important property for developmental robotics, where robots have to be able to learn life-long and continually.

4.2. Experiment 2: Reaching for a Target

In experiment 2, the evaluation task consists of a kinematically simulated arm with $M = 10$ degrees of freedom. The length of each arm segment is 0.6 times the length of the previous segment, and the total length of the arm is 1. The arm should learn to reach for a specific goal [0.0 0.5] with minimal joint angles (expressing a 'comfort' factor), and whilst minimizing acceleration at each time step. Initially, all joint angles are 0, as depicted in Figure 4, and have a null speed. The robot uses PI^2_{CMA} to optimize the shape parameters of a policy, which is described along with the cost function and algorithmic parameters are summarized in Appendix B.

We again evaluate PI^2_{CMA}'s capability to adapt to changing tasks by changing the x-coordinate of the goal for reaching both

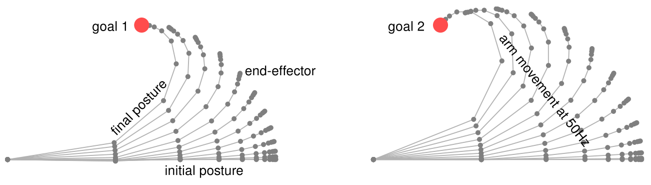

Figure 4. Visualization of the reaching motion (after learning) for 'goal 1' and 'goal 2'

abruptly and gradually, as illustrated in the top graph of Figure 5. First the goal is set to 'goal 1' (cf. Figure 4) and after 150 update to 'goal 2'. Between updates 200 and 250, the x-coordinate of the goal is a sinusoidal, and ends up in 'goal 1' again at update 250.

The middle and bottom graph in Figure 5 depict the learning curves and total exploration magnitude Λ, i.e. the sum over the exploration magnitudes of the individual dimensions: $\Lambda = \sum_{m=1}^{M} \lambda_m$. In the first 30 updates, exploration goes up, which enables fast learning, and consequently the cost goes down rapidly. Between updates 30-100, the exploration decreases, and after 100 updates it approximately reaches its minimum level of $M \cdot \lambda^{\text{init}} = 10 * 0.1 = 1$. Thus, the task has been learned. When the goal changes abruptly at update 150, exploration goes up again. Note that we do not notify the algorithm that the task has changed; the increasing exploration is an emergent property of using probability-weighted averaging to update the covariance matrix. At update 180, the task has again been learned, and exploration is minimal. Whilst the goal is moving, exploration is constantly on, but when the goal remains still again at update 250, it decreases again.

Conclusion

This experiment confirms that PI^2_{CMA} is able to automatically adapt its exploration magnitude to (re)adapt to abruptly or continuously changing tasks.

5. Emergent Proximodistal Maturation

In this experiment, our initial aim was to use the exploration magnitude as a measure of competence to drive the release of degrees of freedom over time, thus using competence progress to adaptively maturing the action space such as proposed in [2]. However, after running some initial experiments we noticed that, without any modification, the PI^2_{CMA} already frees and freezes joints automatically. Therefore maturation appears to be an emergent property of the use of PI^2_{CMA} in such a sensorimotor

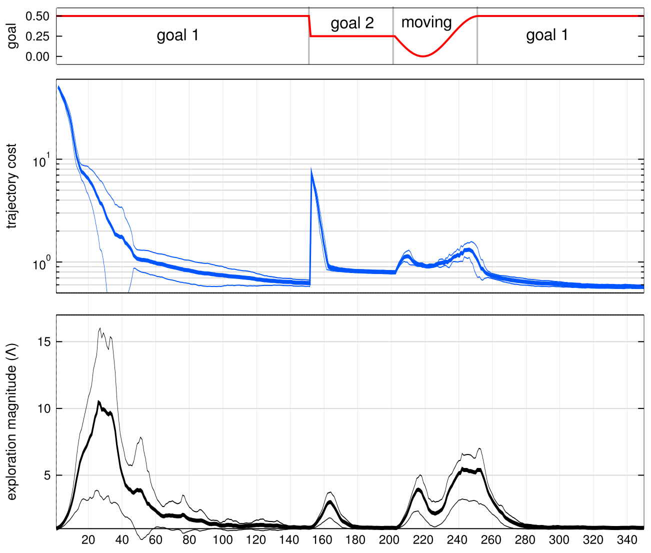

Figure 5. Top: x-coordinate of the task goal. Center: Learning curve ($\mu \pm \sigma$ over 10 learning sessions). Note the logarithmic y-axis. Bottom: Total exploration magnitudes over all joints Λ ($\mu \pm \sigma$ over 10 learning sessions).

space, and there was no need to implement a specific scheme to release degrees of freedom. Rather than conducting a novel experiment, we therefore investigate the first 100 updates of the second experiment in Section 4, in particular the exploration magnitudes of the individual joints $\lambda_{m=1...M}$. These are depicted in Figure 6.

When inspecting the development of the exploration of the different joints λ_m as learning progresses, we notice the following. The exploration magnitude of the first joint λ_1 increases very quickly Ⓐ, i.e. it is freed. After 18 updates λ_1 peaks, and accounts for more than 50% of the total exploration Λ Ⓑ. Then, the second joint is freed and even overtakes the first joint, peaking at update 26 Ⓒ. Subsequently, joint 3 increases, and peaks at update 34 Ⓓ. It thus becomes clear that the first three joints, which have the largest effect on end-effector position, are freed from proximal to more distal ones. At update 50, the goal is reached Ⓔ, and the rest of the learning is concerned with minimizing joint angles and accelerations, which involves all joints. At update 150 the task is learned, and all exploration (beyond λ^{min}) has decayed Ⓕ. Thus, when the task is learned, the exploration in all joints ceases.

Figure 7 plots the movement of the arm during different stages of learning, and visualizes λ_m for each joint as a bar plot. This allows the interpretation of learning in terms of the movement of the arm, and a more direct association between the exploration magnitude of a joint and its position in the arm.

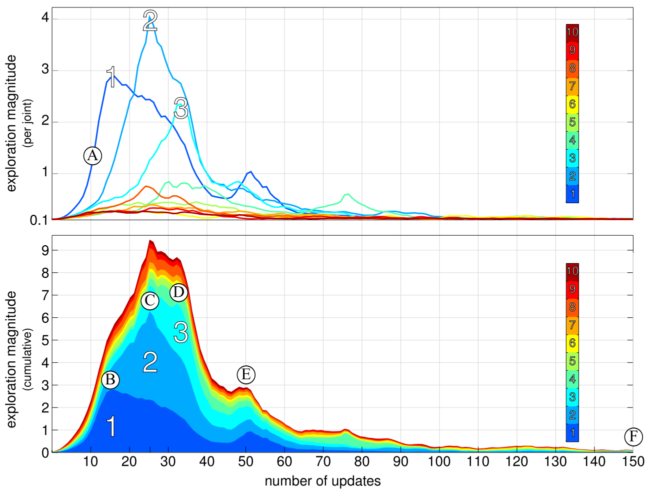

Figure 6. These plots present a closer look at the exploration magnitudes during the first 100 updates of the experiment depicted in Figure 5. Top: Exploration magnitudes λ_m for each joint $m = 1\dots10$ separately. These values are averaged over 10 learning sessions, and thus represents consistent, reproducible behavior. Bottom: The total exploration magnitude Λ, split into the individual components λ_m, i.e. the cumulative of the top graph. In this last graph, $\lambda^{\min} = 0.1$ has been subtracted from all λ_m, as we want to emphasize the exploration *above* the baseline, on which PI^2_{CMA} has an influence. The last graph therefore starts at 0.

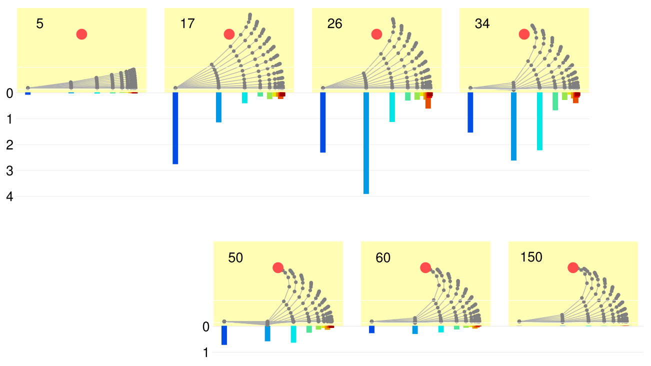

Figure 7. The arm motion at different stages of learning. The numbers next to the arm indicates the number of updates. The exploration magnitude per joint λ_m is plotted as a bar graph below each arm.

5.1. Sensitivity Analysis

In this section, we consider the effect that perturbations of individual joints have on the cost through sensitivity analysis. Sensitivity analysis aims at *"providing an understanding of how the model response variables respond to changes in the input."* [17]. Here, we use sensitivity analysis to investigate how the variation in individual joint angles – the input – influences the variation in the cost – the response variables. This provides a first indication of why proximodistal maturation arises.

In the default posture, all joint angles are zero. This posture is perturbed by setting one of the 10 joint angles to $\frac{\pi}{10}$. The 10 possible perturbations, one for each joint, are visualized in Figure 8. For the default and perturbed configuration, we then compute the distance of the end-effector to the target $||\mathbf{x}_{t_N} - \mathbf{x}^g||$. Because this is a static context there are no joint accelerations, and the immediate costs (4) are not included. The right graph plots the difference in cost between the outstretched arm, and the slightly bent arm (where one joint angle is $\frac{\pi}{10}$).

For all arm configurations, we see that proximal joints lead to a higher average difference in the distance to the target than more distal ones. This should not come as a surprise, as rotating more proximal joints leads to smaller movement in the end-effector space, and it is the end-effector space that determines the distance to the target. As a consequence, the same magnitude of perturbation will lead to a larger difference in cost for more proximal joints.

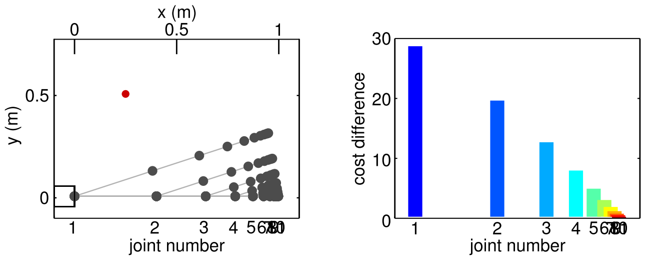

Figure 8. Results of the sensitivity analysis.

The goal of PI^2_{CMA} is to minimize costs through exploring and updating in parameter space. The results in Figure 8 demonstrate that perturbing proximal joints leads to larger differences in costs than distal joints. Therefore, an optimizer can be expected to minimize costs more quickly if it initially focuses exploration on proximal joints, rather than distal ones.

Conclusion

PI^2_{CMA} freezes and frees joint sequentially, depending on where the robot is on its self-organized developmental trajectory to learn the task. As learning progresses, joints are freed in a proximal-to-distal order, as is observed when infants learn to reach [3, 11]. Rather than having to specify the order of freeing/freezing joints [2], and/or their timing [13], structured maturation is an emergent property of probability-weighted covariance matrix updating in the PI^2_{CMA} algorithm. The sensitivity analysis provides a partial explanation of why this behavior arises.

In our current work, we are studying the robustness of the emergence of proximo-distal freeing of DOFs to different body structures (for instance human link lengths vs. equidistant link lengths), as well as the location of the target within the workspace of the arm. Our preliminary results show that longer relative proximal link lengths lead to a more pronounced proximo-distal freeing of joints.

6. Conclusion

In this article, we demonstrate that $\mathrm{PI}^2_{\mathrm{CMA}}$ shows useful developmental properties for adaptive exploration and life-long learning of motor skills. First, we demonstrated how it continually and automatically adapts to abruptly or continuously changing tasks, and without direct external information about these changes. Second, we show how the proximodistal maturation observed in humans [3, 11], and previously demonstrated to be highly useful for robot learning in high-dimensions [2], was here entirely self-organized. Identifying the detailed roles of body structure, target reachability, learning algorithm and their coupling for maturational self-organization will thus be a focus of future work.

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] A. Baranes and P-Y. Oudeyer. The interaction of maturational constraints and intrinsic motivations in active motor development. In *IEEE International Conference on Development and Learning*, 2011.

[3] N. E. Berthier, R.K. Clifton, D.D. McCall, and D.J. Robin. Proximodistal structure of early reaching in human infants. *Exp Brain Res*, 1999.

[4] L. Berthouze and M. Lungarella. Motor skill acquisition under environmental perturbations: On the necessity of alternate freezing and freeing degrees of freedom. *Adaptive Behavior*, 12(1):47–63, 2004.

[5] Josh C. Bongard. Morphological change in machines accelerates the evolution of robust behavior. *Proceedigns of the National Academy of Sciences of the United States of America (PNAS)*, January 2010.

[6] Ronen I. Brafman and Moshe Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, March 2003. ISSN 1532-4435.

[7] T. Glasmachers, T. Schaul, S. Yi, D. Wierstra, and J. Schmidhuber. Exponential natural evolution strategies. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 393–400. ACM, 2010.

[8] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[9] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[10] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Mach. Learn.*, 49 (2-3):209–232, 2002. ISSN 0885-6125.

[11] J. Konczak, M. Borutta, T Helge, and J. Dichgans. The development of goal-directed reaching in infants: hand trajectory formation and joint torque control. *Experimental Brain Research*, 1995.

[12] A. Miyamae, Y. Nagata, I. Ono, and S. Kobayashi. Natural policy gradient methods with parameter-based exploration for control tasks. *Advances in Neural Information Processing Systems*, 2:437–441, 2010.

[13] Y. Nagai, M. Asada, and K. Hosoda. Learning for joint attention helped by functional development. *Advanced Robotic*, 20(10), 2006.

[14] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[15] R. Ros and N. Hansen. A Simple Modification in CMA-ES Achieving Linear Time and Space Complexity. In *Proceedings on Parallel Problem Solving from Nature*, 296–305, 2008.

[16] Thomas Rückstiess, Frank Sehnke, Tom Schaul, Daan Wierstra, Yi Sun, and Jürgen Schmidhuber. Exploring parameter space in reinforcement learning. *Paladyn. Journal of Behavioral Robotics*, 1:14–24, 2010. ISSN 2080-9778.

[17] A. Saltelli, K. Chan, and E. M. Scott. *Sensitivity analysis*. Chichester: Wiley, 2000.

[18] Stefan Schaal. The sl simulation and real-time control software package. Technical report, University of Southern California, 2007.

[19] Matthew Schlesinger, Domenico Parisi, and Jonas Langer. Learning to reach by constraining the movement search space. *Developmental Science*, 3:67–80, 2000.

[20] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[21] Andrew Stout, George D. Konidaris, and Andrew G. Barto. Intrinsically motivated reinforcement learning: A promising framework for developmental robot learning. In *AAAI*, 2005.

[22] Freek Stulp. Adaptive exploration for continual reinforcement learning. In *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[23] Freek Stulp and Pierre-Yves Oudeyer. Emergent proximo-distal maturation through adaptive exploration. In *Inter-*

national Conference on Development and Learning (ICDL), 2012. Paper of Excellence Award.

[24] Freek Stulp and Olivier Sigaud. Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

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

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

[27] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Appendix

A. Experiment 1: T-Ball

Cost function.

The cost function for this task is:

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

$$\phi = \begin{cases} \text{if in target area} & 0 \\ \text{else} & \text{distance to target area in m} \end{cases} \tag{2}$$

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 Figure 2.

Policy Representation

In this task, the policy is represented as a Dynamic Movement Primitive [9]. The DMP has 7 dimensions to control the 7 joint angles of the arm. Each dimension has $B = 3$ basis functions, and is initialized as a minimum-jerk trajectory of duration 1s from the start to the end pose as visualized in Figure 2.

PI^2_{CMA} parameters

The PI^2_{CMA} parameters are $K = 20$, $h = 10$. Initially $\mathbf{\Sigma}^{\text{init}} = \lambda^{\text{init}}\mathbf{I}_3$ with λ^{init}=20, and λ^{min}=0.02.

B. Experiment 2: Target Reaching

Cost function

The terminal costs of this task are expressed in (3), where $||\mathbf{x}_{t_N} - g||$ represented the distance between the 2-D Cartesian coordinates of the end-effector ($\mathbf{x}_{t_N}$) and the goal $g_1 = [0.0\ 0.5]$ or $g_2 = [0.0\ 0.25]$ at the end of the movement at t_N. The terminal cost also penalizes the joint with the largest angle at max($\mathbf{q}_{t_N}$), expressing a comfort effect, with maximum comfort being the initial position. The immediate costs at each time step r_t in (4) penalize joint accelerations. The weighting term $(M + 1 - m)$ penalizes DOFs closer to the origin, the underlying motivation being that wrist movements are less costly than shoulder movements for humans, cf. [25]. This cost term was taken from [25]. In the context of this article, it cannot be the reason for the proximodistal maturation in Section 5. Rather than favoring a proximodistal maturation, this cost term works *against* it, as proximal joints are penalized *more* for the accelerations that arise due to exploration.

$$\phi_{t_N} = 10^4 ||\mathbf{x}_{t_N} - g||^2 + \max(\mathbf{q}_{t_N}) \qquad \text{Terminal cost} \tag{3}$$

$$r_t = 10^{-5} \frac{\sum_{m=1}^{M}(M+1-m)(\ddot{q}_{t,m})^2}{\sum_{m=1}^{M}(M+1-m)} \qquad \text{Immediate cost} \tag{4}$$

Policy representation

The acceleration $\ddot{q}_{m,t}$ of the m^{th} joint at time t is determined as a linear combination of basis functions, where the parameter vector $\boldsymbol{\theta}_m$ represents the weighting of joint m.

$$\ddot{q}_{m,t} = \mathbf{g}_t^{\intercal}\boldsymbol{\theta}_m \qquad \text{Acc. of joint } m \tag{5}$$

$$[\mathbf{g}_t]_b = \frac{\Psi_b(t)}{\sum_{b=1}^{B}\Psi_b(t)} \qquad \text{Basis functions} \tag{6}$$

$$\Psi_b(t) = \exp\left(-(t-c_b)^2/w^2\right) \qquad \text{Kernel} \tag{7}$$

The centers $c_{b=1...B}$ of the $B = 3$ kernels Ψ are spaced equidistantly in the 0.5s duration of the movement, and all have a width of $w = 0.05s$. Since we do not simulate arm dynamics, the joint velocities and angles are acquired by integrating the accelerations.

PI^2_{CMA} parameters

The input parameters of PI^2_{CMA} are set as follows. The initial parameter vector is $\theta_\mu = \mathbf{0}$, which means the arm is completely stretched, and not moving at all over time. The number of trials per update is $K = 10$, and the eliteness parameter is $h = 10$

(the default values suggested by [25]). The initial and minimum exploration magnitude of each joint m is set to $\lambda_m^{\text{init}} = \lambda_m^{\text{min}} = 0.1$, unless stated otherwise.

For this task, we use an extended version of PI^2_{CMA}, that also uses 'evolution paths' to update the covariance matrix, as in CMA-ES. Since this is not part of the core algorithm, we refer to equations (14)-(17) in Hansen et al. [8] or equations (20)-(23) in Stulp et al. [24] for the implementation of evolution paths. The evolution paths effectively act as a low-pass filter on the covariance matrix update. From other results not reported here due to space limitations, we conclude that the use of evolution paths has only a very marginal impact on this particular task, and does not influence the emergence of proximodistal maturation.