HAL open science

# Policy Improvement Methods: Between Black-Box Optimization and Episodic Reinforcement Learning

Freek Stulp, Olivier Sigaud

▶ **To cite this version:**

Freek Stulp, Olivier Sigaud. Policy Improvement Methods: Between Black-Box Optimization and Episodic Reinforcement Learning. 2012. hal-00738463

**HAL Id: hal-00738463**

**https://hal.science/hal-00738463**

Preprint submitted on 4 Oct 2012

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Policy Improvement Methods: Between Black-Box Optimization and Episodic Reinforcement Learning

Freek Stulp[*][†] and Olivier Sigaud[‡]

**Abstract**

Policy improvement methods seek to optimize the parameters of a policy with respect to a utility function. There are two main approaches to performing this optimization: reinforcement learning (RL) and black-box optimization (BBO). Whereas BBO algorithms are generic optimization methods that, due to there generality, may also be applied to optimizing policy parameters, RL algorithms are specifically tailored to leveraging the structure of policy improvement problems. In recent years, benchmark comparisons between RL and BBO have been made, and there has been several attempts to specify which approach works best for which types of problem classes.

In this article, we make several contributions to this line of research: 1) We define four algorithmic properties that further clarify the relationship between RL and BBO: action-perturbation vs. parameter-perturbation, gradient estimation vs. reward-weighted averaging, use of only rewards vs. use of rewards *and* state information, actor-critic vs. direct policy search. 2) We show how the chronology of the derivation of ever more powerful algorithms displays a trend towards algorithms based on parameter-perturbation and reward-weighted averaging. A striking feature of this trend is that it has moved RL methods closer and closer to BBO. 3) We continue this trend by applying two modifications to the state-of-the-art "Policy Improvement with Path Integrals" ($\text{PI}^2$), which yields an algorithm we denote $\text{PI}^{\text{BB}}$. We show that $\text{PI}^{\text{BB}}$ is a BBO algorithm, and, more specifically, that it is a special case of the "Covariance Matrix Adaptation – Evolutionary Strategy" algorithm. Our empirical evaluation demonstrates that the simpler $\text{PI}^{\text{BB}}$ outperforms $\text{PI}^2$ on simple evaluation tasks in terms of convergence speed and final cost. 4) Although our evaluation implies that, for these five tasks, BBO outperforms RL, we do not hold this to be a general statement, and provide an analysis of why these tasks are particularly well-suited for BBO. Thus, rather than making the case for BBO or RL, one of the main contributions of this article is rather to provide an algorithmic framework in which such cases may be made, as $\text{PI}^{\text{BB}}$ and $\text{PI}^2$ use identical perturbation and parameter update methods, and differ only in being BBO and RL approaches respectively.

## 1 Introduction

Over the last two decades, the convergence speed and robustness of policy improvement methods has increased dramatically, such that they are now able to learn a variety of challenging robotic tasks (Theodorou et al., 2010; Rückstiess et al., 2010b; Tamosiumaite et al., 2011; Kober and Peters, 2011; Buchli et al., 2011; Stulp et al., 2012). Several underlying trends have accompanied this performance increase. The first is related to exploration, where there has been a transition from *action perturbing* methods, which perturb the output of the policy at each time step, to *parameter perturbing* methods, which perturb the

[*]Robotics and Computer Vision, ENSTA-ParisTech, Paris

[†]FLOWERS Research Team, INRIA Bordeaux Sud-Ouest, Talence, France

[‡]Institut des Systèmes Intelligents et de Robotique, Université Pierre Marie Curie CNRS UMR 7222, Paris

parameters of the policy itself (Rückstiess et al., 2010b). The second trend pertains to the parameter update, which has moved from *gradient-based* methods towards updates based on *reward-weighted averaging* (Stulp and Sigaud, 2012).

A striking feature of these trends, visualized in Figure 1, and described in detail in Section 2, is that they have moved reinforcement learning (RL) approaches to policy improvement closer and closer to black-box optimization (BBO). This class of algorithms is depicted in the right-most column of Figure 1. In fact, two state-of-the-art algorithms that have been applied to policy improvement — $\mathrm{PI}^2$ (Theodorou et al., 2010) and CMA-ES (Hansen and Ostermeier, 2001) — are so similar that a line-by-line comparison of the algorithms is feasible (Stulp and Sigaud, 2012). The main difference is that whereas $\mathrm{PI}^2$ is an RL algorithm — it uses information about rewards received at each time step *during* exploratory policy executions — whereas CMA-ES is a BBO algorithm — it uses only the *total* reward received during execution, which enables it to treat the utility function as a black box that returns one scalar value.

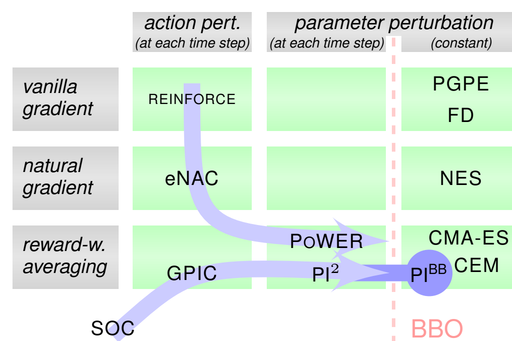


Figure 1: Classification of policy improvement algorithms. The vertical dimension categorizes the update method used, and the horizontal dimension the method used to perturb the policy. The two streams represent both the derivation history of policy improvement algorithms. The algorithms are discussed in Section 2.

In this article, we make the relation between RL and BBO even more explicit by taking these trends one (ultimate) step further. We do so by introducing $\mathrm{PI}^{\mathrm{BB}}$ (in Section 3), which simplifies the exploration and parameter update methods of $\mathrm{PI}^2$. These modifications are consistent with $\mathrm{PI}^2$'s derivation from stochastic optimal control. An important insight is that $\mathrm{PI}^{\mathrm{BB}}$ is actually a BBO algorithm, as discussed in Section 4.1. More specifically, we show that $\mathrm{PI}^{\mathrm{BB}}$ is a special case of CMA-ES. One of the main contributions of this article is thus to draw an explicit bridge from RL to BBO approaches to policy improvement, as visualized by the dark line from $\mathrm{PI}^2$ to $\mathrm{PI}^{\mathrm{BB}}$ in Figure 1.

We thus have a pair of algorithms — $\mathrm{PI}^2$ and $\mathrm{PI}^{\mathrm{BB}}$— that use the same method for exploration (parameter perturbation) and parameter updating (reward-weighted averaging), and differ only in being RL ($\mathrm{PI}^2$) or BBO ($\mathrm{PI}^{\mathrm{BB}}$) approaches to policy improvement. This allows for a more objective comparison of RL/BBO algorithms than, for instance, comparing eNAC and CMA-ES (Heidrich-Meisner and Igel, 2008a; Rückstiess et al., 2010b), as eNAC is an action-perturbing, gradient-based RL algorithm, and CMA-ES is a parameter-perturbing, reward-weighted averaging BBO algorithm. If one is found to outperform the other on a particular task, does it do so due to the different parameter update methods? Or is it due to the difference in the policy perturbation? Or because one is an RL method and the other BBO? Using the $\mathrm{PI}^2/\mathrm{PI}^{\mathrm{BB}}$ pair allows us to specifically investigate the latter question, whilst keeping the other algorithmic features the same.

The $\mathrm{PI}^2/\mathrm{PI}^{\mathrm{BB}}$ pair may thus be the key to providing "[s]trong empirical evidence for the power of evolutionary RL and convincing arguments why certain evolutionary algorithms are particularly well suited for certain RL problem classes" (Heidrich-Meisner and Igel, 2008a), and could help verify or falsify the five conjectures proposed by Togelius et al. (2009, Section 4.1), about which types of problems are particularly suited for RL and BBO

approaches to policy improvement. As a first step in this direction, we compare the performance of $PI^2$ and $PI^{BB}$ in terms of convergence speed and final cost on the evaluation tasks from (Theodorou et al., 2010) in Section 3. Although $PI^{BB}$ has equal or better performance than $PI^2$ on these tasks, our aim in this article is not to make a case for either RL or BBO approaches to policy improvement — in general we expect the most appropriate method to vary from task to task, as discussed in Section 5 — but rather to provide a pair of algorithms that allow such targeted comparisons in the first place.

In summary, the main contributions of this article are:

- Providing an overview and classification of policy improvement algorithms.
- Deriving $PI^{BB}$ by simplifying the perturbation and update methods of $PI^2$.
- Empirically comparing $PI^2$ and $PI^{BB}$ on the five tasks proposed by Theodorou et al. (2010), and showing that $PI^{BB}$ has equal or superior performance.
- Demonstrating that $PI^{BB}$ is a BBO algorithm. In particular, it is a special case of CMA-ES.
- Providing an algorithmic pair ($PI^2$ and $PI^{BB}$) with which it is easier to verify the conjectures proposed by Togelius et al. (2009).

The rest of this article is structured as follows. In the next section, we describe the policy improvement algorithms depicted in Figure 1, explain their key differences, and classify them according to these differences. In Section 3, we show how $PI^{BB}$ is derived by applying two simplifications to $PI^2$, and we compare the algorithms empirically on five tasks. The $PI^{BB}$ algorithm is analyzed more closely in Section 4; in particular, we show that $PI^{BB}$ is a BBO algorithm, and discuss several reasons why it outperforms $PI^2$ on the tasks used. In Section 6 we summarize the main contributions of the article, and present future research opportunities instigated by this article.

## 2 Background

In RL, the policy $\pi$ maps states to actions. The optimal policy $\pi^*$ chooses the action that optimizes the cumulative discounted reward over time. When the state and actions sets of the system are discrete, finding the optimal policy $\pi^*$ can be cast in the framework of discrete Markov Decision Processes (MDPs) and solved with Dynamic Programming (DP) or RL methods (Sutton and Barto, 1998). For problems where the state is continuous, many state approximation techniques exist in the field of Approximate Dynamic Programming (ADP) methods (Powell, 2007). But when the action space also becomes continuous, the extension of DP or ADP methods results in optimization problems that have proven hard to solve in practice (Santamaría et al., 1997).

In such contexts, a policy cannot be represented by enumerating all actions, so parametric policy representations $\pi_{\boldsymbol{\theta}}$ are required, where $\boldsymbol{\theta}$ is a vector of parameters. Thus, finding the optimal policy $\pi^*$ corresponds to finding the optimal policy parameters $\boldsymbol{\theta}^*$, i.e. those that maximize cumulative discounted reward. As finding the $\boldsymbol{\theta}$ corresponding to the global optimum is generally too expensive, policy improvement methods are local methods that rather search for a local optimum of the expected reward.

In episodic RL, on which this article focusses, the learner executes a task until a terminal state is reached. Executing a policy from an initial state until the terminal state, called a "roll-out", leads to a trajectory $\boldsymbol{\tau}$, which contains information about the states visited, actions executed, and rewards received. Many policy improvements use an iterative process of exploration, where the policy is executed $K$ times leading to trajectories $\boldsymbol{\tau}_{k=1...K}$, and

parameter updating, where the policy parameters $\boldsymbol{\theta}$ are updated based on this batch of trajectories. This process is explained in more detail in the generic policy improvement loop in Figure 2.

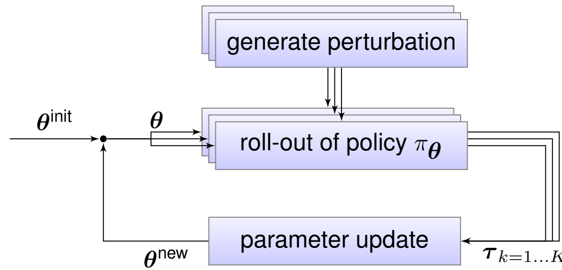


Figure 2: Generic policy improvement loop. In each iteration, the policy is executed $K$ times. One execution of a policy is called a 'Monte Carlo roll-out', or simply 'roll-out'. Because the policy is perturbed (different perturbation methods are described in Section 2.1.3), each execution leads to slightly different trajectories in state/action space, and potentially different rewards. The exploration phase thus leads to a set of different trajectories $\boldsymbol{\tau}_{k=1\ldots K}$. Based on these trajectories, policy improvement methods then update the parameter vector $\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}^{\text{new}}$ such that the policy is expected to incur lower costs/higher rewards. The process then continues with the new $\boldsymbol{\theta}^{\text{new}}$ as the basis for exploration.

In this section, we give an overview of algorithms that implement this loop. We distinguish between three main classes of algorithms, based on whether their derivation is based mainly — they are not mutually exclusive — on principles based on lower bounds on the expected return (Section 2.1), path integral stochastic optimal control (Section 2.2) or BBO (Section 2.3).

For each algorithm, we make a 'fact sheet', in which the following questions are answered:

Fact sheet (to be filled in by all algorithms)

**Perturbation:** How is the policy perturbed?
**Trajectory:** What information must be stored in the trajectory resulting from the execution of the policy?
**Actor-Critic:** Is the method an actor-critic method, or a direct policy search method?
**Update:** How are the parameters updated?

## 2.1 Policy Improvement through Lower Bounds on the Expected Return

We now briefly describe three algorithms that build on one another to achieve ever more powerful policy improvement methods, being REINFORCE (Williams, 1992), eNAC (Peters and Schaal, 2008b), and POWER (Kober and Peters, 2011). All these algorithms may be derived from a common framework based on the lower bound on the expected return, as demonstrated by Kober and Peters (2011). In this section, we focus on properties of the resulting algorithms, rather than on their derivations.

Our main aim here is to use a set of known algorithms to answer the questions in the fact sheet, thus providing a basis for considering the questions in perhaps unfamiliar and more intricate contexts in Section 2.2 and 3. Since these algorithms have already been covered in extensive surveys (Peters and Schaal, 2008b, 2007; Kober and Peters, 2011), we do not present them in full detail here, as this does not serve the particular aim of this section.

An underlying assumption of the algorithms presented in Section 2.1 and 2.2 is that the policies are represented as $\mathbf{u}_t = \mathbf{g}(\mathbf{x}, t)^{\intercal}\boldsymbol{\theta}$; $\mathbf{g}$ is a set of basis functions, for instance Gaussian kernels, $\boldsymbol{\theta}$ are the policy parameters, $\mathbf{x}$ is the state, and $t$ is time since the roll-out started.

### 2.1.1 REINFORCE

The REINFORCE algorithm (Williams, 1992) ("reward increment = nonnegative factor $\times$ offset reinforcement $\times$ characteristic eligibility") uses a stochastic policy to foster exploration (1), where $\pi_{\boldsymbol{\theta}}(\mathbf{x})$ returns the nominal motor command[1], and $\boldsymbol{\epsilon}_t$ is a perturbation of this command at time $t$. In REINFORCE, this policy is executed $K$ times with the same $\boldsymbol{\theta}$, and the states/actions/rewards that result from a roll-out are stored in a trajectory.

Given $K$ such trajectories, the parameters $\boldsymbol{\theta}$ are then updated by first estimating the gradient $\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ (2) of the expected return $J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{i=1}^{N} r_{t_i} | \pi_{\boldsymbol{\theta}}\right]$. Here, the trajectories are assumed to be of equal length, i.e. having $N$ discrete time steps $t_{i=1\ldots N}$. The notation in (2) estimates the gradient $\hat{\nabla}_{\boldsymbol{\theta}_d} J(\boldsymbol{\theta})$ for each parameter entry $d$ in the vector $\boldsymbol{\theta}$ separately. Riedmiller et al. (2007) provides a concise yet clear explanation how to derive (2). The baseline (3) is chosen so that it minimizes the variation in the gradient estimate (Peters and Schaal, 2008a). Finally, the parameters are updated through steepest gradient ascent (4), where the open parameter $\alpha$ is a learning rate.

**Policy perturbation during a roll-out**

$$\mathbf{u}_t = \pi_{\boldsymbol{\theta}}(\mathbf{x}) + \boldsymbol{\epsilon}_t \tag{1}$$

**Parameter update, given $K$ roll-outs**

$$\hat{\nabla}_{\boldsymbol{\theta}_d} J(\boldsymbol{\theta}) = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{i} \nabla_{\boldsymbol{\theta}_d} \log \pi(\mathbf{u}_{t_j,k} | \mathbf{x}_{t_j,k})(r_{t_i,k} - b_{t_i}^d) \tag{2}$$

$$b_{t_i}^d = \frac{\sum_{k=1}^{K} \sum_{j=1}^{i} \left(\nabla_{\boldsymbol{\theta}_d} \log \pi(\mathbf{u}_{t_j,k} | \mathbf{x}_{t_j,k})\right)^2 r_{t_i,k}}{\sum_{k=1}^{K} \sum_{j=1}^{i} \left(\nabla_{\boldsymbol{\theta}_d} \log \pi(\mathbf{u}_{t_j,k} | \mathbf{x}_{t_j,k})\right)^2} \tag{3}$$

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta} + \alpha \hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{4}$$

Fact sheet for REINFORCE

**Perturbation:** A stochastic policy is used, i.e. the output of the nominal policy is perturbed, cf. (1).
**Trajectory:** To compute (2) and (3), the trajectory needs to contain, for each time step $i$, the reward $r_{t_i}$, as well as the state $x_{t_i}$ and action $u_{t_i}$, as they are required to compute $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_t|\mathbf{x}_t)$.
**Actor-Critic:** The value function is not approximated, so it is a direct policy search method.
**Update:** The update is based on the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, cf. (4). Note that in order to compute this gradient, the policy must be differentiable w.r.t. its parameters: $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_t|\mathbf{x}_t)$.

### 2.1.2 eNAC

One issue with REINFORCE is that it requires many roll-outs for one parameter update, and the resulting trajectories cannot be reused for later updates. This is because we need to perform a roll-out each time we want to compute $\sum_{i=1}^{N}[\ldots]r_{t_i}$ in (2). Such methods are known as 'direct policy search' methods. Actor-critic methods, such as "Episodic Natural Actor Critic" (eNAC), address this issues by using a value function $V_{\pi_{\boldsymbol{\theta}}}$ as a more compact representation of long-term reward than sample episode $R(\boldsymbol{\tau})$, allowing them to make more efficient use of samples.

In continuous state-action spaces, $V_{\pi_{\boldsymbol{\theta}}}$ cannot be represented exactly, but must be estimated from data. Actor-critic methods therefore update the parameters in two steps: 1) approximate the value function from the point-wise estimates of the cumulative rewards observed in the trajectories acquired from roll-outs of the policy; 2) update the parameters

[1] With this notation, the policy $\pi_{\boldsymbol{\theta}}(\mathbf{x})$ is actually deterministic. A truly stochastic policy is denoted as $\mathbf{u}_t \sim \pi_{\boldsymbol{\theta}}(\mathbf{u}|\mathbf{x}) = \mu(\mathbf{x}) + \boldsymbol{\epsilon}_t$ (Riedmiller et al., 2007), where $\mu(\mathbf{x})$ is a deterministic policy that returns the nominal command. We use our notation for consistency with parameter perturbation, introduced in Section 2.1.3. For now, it is best to consider the sum $\pi_{\boldsymbol{\theta}}(\mathbf{x}) + \boldsymbol{\epsilon}_t$ to be the stochastic policy, rather than just $\pi_{\boldsymbol{\theta}}(\mathbf{x})$.

using the value function. In contrast, direct policy search updates the parameters directly using point-wise estimates[2], as visualized in Figure 3. The main advantage of having a value function is that it generalizes; whereas $K$ roll-outs provide only $K$ point-wise estimates of the cumulative reward, a value function approximated from these $K$ point-wise estimates is also able to provide estimates not observed in the roll-outs.

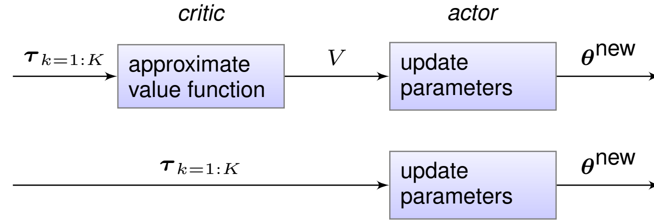


Figure 3: Actor-critic (above) and direct policy search (below).

Another issue is that in REINFORCE the 'naive', or 'vanilla'[3], gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is sensitive to different scales in parameters. To find the true direction of steepest descent towards the optimum, independent of the parameter scaling, eNAC uses the Fischer information matrix $F$ to determine the 'natural gradient': $\boldsymbol{\theta}^{new} = \boldsymbol{\theta} + \alpha F^{-1}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. In practice, the Fischer information matrix need not be computed explicitly (Peters and Schaal, 2008b).

Fact sheet for eNAC

**Perturbation** and **Trajectory:** Same as for REINFORCE.
**Actor-critic:** As the name implies, eNAC is an actor-critic approach, because it first approximates the value function with LSTD(1) (critic), and then uses the value function to update the policy parameters (actor).
**Update:** eNAC uses the natural gradient to update the policy parameters.

Thus, going from REINFORCE to eNAC represents a transition from direct policy search to actor-critic, and from vanilla to natural gradients.

#### 2.1.3 PoWER

REINFORCE and eNAC are both 'action perturbing' methods which perturb the nominal command at each time step $\mathbf{u}_t = \mathbf{u}_t^{\text{nominal}} + \boldsymbol{\epsilon}_t$, cf. (1). Action-perturbing algorithms have several disadvantages: 1) Samples are drawn independently from one another at each time step, which leads to a very noisy trajectory in action space (Rückstiess et al., 2010b). 2) Consecutive perturbations may cancel each other and are thus washed out (Kober and Peters, 2011). The system also often acts as a low-pass filter, which further reduces the effects of perturbations that change with a high frequency. 3) On robots, high-frequency changes in actions, for instance when actions represent motor torques, may lead to dangerous behavior, or damage to the robot (Rückstiess et al., 2010b). 4) It causes a large variance in parameter updates, an effect which grows with the number of time steps (Kober and Peters, 2011).

The "Policy Learning by Weighting Exploration with the Returns" (PoWER) algorithm therefore implements a different policy perturbation scheme first proposed by Rückstiess et al. (2010a), where the parameters $\boldsymbol{\theta}$ of the policy, rather than its output, are perturbed, i.e. $\pi_{\boldsymbol{\theta}+\boldsymbol{\epsilon}_t}(\mathbf{x})$ rather than $\pi_{\boldsymbol{\theta}}(\mathbf{x}) + \boldsymbol{\epsilon}_t$. This distinction has been illustrated in Figure 4.

[2]The point-wise estimates are sometimes considered to be a special type of critic; in this article we use the term 'critic' only when it is a function approximator.

[3]'Vanilla' refers to the canonical version of an entity. The origin of this expression lies in ice cream flavors; i.e. 'plain vanilla' vs. 'non-plain' flavors, such as strawberry, chocolate, etc.

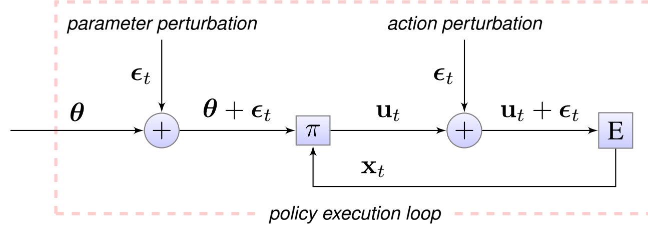


Figure 4: Illustration of action and policy parameter perturbation. Action perturbation is applied to the *output* $\mathbf{u}_t$ of the policy, whereas policy parameter perturbation is applied to the *parameters* $\boldsymbol{\theta}$ of the policy. The perturbations are sampled at each time step, inside the loop in which the policy is executed.

REINFORCE and eNAC estimate gradients, which is not robust when noisy, discontinuous utility functions are involved. Furthermore, they require the manual tuning of the learning rate $\alpha$, which is not straight-forward, but critical to the performance of the algorithms (Theodorou et al., 2010; Kober and Peters, 2011). The PoWER algorithm proposed by Kober and Peters (2011) addresses these issues by using reward-weighted averaging, which rather takes a weighted average of a set of $K$ exploration vectors $\boldsymbol{\epsilon}_{k=1...K}$ as follows:

**Policy perturbation during a roll-out**

$$\mathbf{u}_t = \pi_{\boldsymbol{\theta}+\boldsymbol{\epsilon}_t}(\mathbf{x}), \text{ with } \boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \boldsymbol{\Sigma}) \tag{5}$$

**Parameter update, given $K$ roll-outs**

$$S^k_{t_i} = \textstyle\sum_{j=i}^{N} r^k_j \tag{6}$$

$$\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta} + \left(\mathrm{E}\left[\sum_{i=1}^{N} \mathbf{W} S_{t_i}\right]\right)^{-1} \left(\mathrm{E}\left[\sum_{i=1}^{N} \mathbf{W}\boldsymbol{\epsilon}_{t_i} S_{t_i}\right]\right) \tag{7}$$

$$\approx \boldsymbol{\theta} + \left(\sum_{k=1}^{K}\sum_{i=1}^{N} \mathbf{W}_{t_i} S^k_{t_i}\right)^{-1} \left(\sum_{k=1}^{K}\sum_{i=1}^{N} \mathbf{W}_{t_i}\boldsymbol{\epsilon}^k_{t_i} S^k_{t_i}\right) \tag{8}$$

$$\text{with } \mathbf{W}_{t_i} = \mathbf{g}_{t_i}\mathbf{g}_{t_i}^{\intercal}(\mathbf{g}_{t_i}^{\intercal}\boldsymbol{\Sigma}\mathbf{g}_{t_i})^{-1}$$

where $K$ refers to the number of roll-outs, and $\mathbf{g}_{t_i}$ is a vector of the basis function activations at time $t_i$. The update (8) may be interpreted as taking the average of the perturbation vectors $\boldsymbol{\epsilon}_k$, but weighting them with $S_k / \sum_{l=1}^{K} S_l$, which is a normalized version of the reward-to-go $S_k$. Hence the name reward-weighted averaging. An important property of reward-weighted averaging is that it follows the natural gradient (Arnold et al., 2011), *without* having to actually compute the gradient or the Fischer information matrix. This leads to more robust updates.

The final main difference between REINFORCE/eNAC and PoWER is that the former require roll-out trajectories to contain information about the state and actions, as they must compute $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_t|\mathbf{x}_t)$ to perform an update. In contrast, PoWER only uses information about the rewards $r_t$ from the trajectory, as these are necessary to compute the expected return (6). The basis function $\mathbf{g}$ are parameters of the algorithm, and must not be stored in the trajectory.

Fact sheet for PoWER

**Perturbation:** The parameters of the policy are perturbed ($\boldsymbol{\theta} + \boldsymbol{\epsilon}_t$), rather than the output of the policy, cf. (5).
**Trajectory:** To determine the expected return (6), the trajectory must contain the reward received at each time step $r_{t_{i=1...N}}$. The state and actions are not needed to perform an update.
**Actor-Critic:** The value function is not approximated, so it is a direct policy search method.
**Update:** The update is based on reward-weighted averaging (8), rather than estimating a gradient.

In summary, going from eNAC to POWER represents a transition from action perturbation to policy parameter perturbation, from estimating the gradient to reward-weighted averaging, and from actor-critic back to direct policy search (as in REINFORCE).

## 2.2 Policy Improvement with Path Integrals

In this section, we describe the second stream (SOC→GPIC→PI$^2$) in Figure 1. The aim of this section is to lay the foundation for Section 3, in which we analyze how the transition from action perturbation to policy parameter perturbation is made within the PI$^2$ derivation. We only explain those parts of the derivation that serve this aim; for the complete PI$^2$ derivation on which this section is based, we refer to Theodorou et al. (2010). Note that in the previous section, algorithms aimed at maximizing *rewards*. In optimal control, the convention is rather to define *costs*, which should be minimized.

### 2.2.1 Source: Stochastic Optimal Control

The PI$^2$ derivation is based on stochastic optimal control (SOC). Note that SOC in itself does not involve parameterized policies, exploration or learning, and is not an algorithm. It is rather the definition of a domain in which, given a model of the control system and a cost function, a set of equations (Hamilton Jacobi Bellman) is derived which must be solved in order to compute the optimal controls. In Section 2.2.2, we present a path-integral-based solution to this problem formulation, and in Section 2.2.3 we apply this solution to parameterized policies, which yields the PI$^2$ algorithm.

In SOC, the dynamics of the control system is assumed to take the following form:

$$\dot{\mathbf{x}}_t = \mathbf{f}(\mathbf{x}_t) + \mathbf{G}(\mathbf{x}_t)\,(\mathbf{u}_t + \epsilon_t) = \mathbf{f}_t + \mathbf{G}_t\,(\mathbf{u}_t + \epsilon_t) \tag{9}$$

where $\mathbf{x}_t$ denotes the state of the system, $\mathbf{G}_t = \mathbf{G}(\mathbf{x}_t)$ the control matrix, $\mathbf{f}_t = \mathbf{f}(\mathbf{x}_t)$ the passive dynamics, $\mathbf{u}_t$ the control vector and $\epsilon_t$ zero-mean Gaussian noise with covariance $\boldsymbol{\Sigma}$.

In the system model of SOC, we see that actions are perturbed, because of $\mathbf{u}_t + \epsilon_t$ (9). However, the nature of this perturbation is quite different to those in the algorithms in Section 2. In SOC, these perturbations represent additive motor stochasticity that arises when applying the command $\mathbf{u}_t$ to the system, for instance due to imperfectly calibrated motors or wear-and-tear of the system. This is quite distinct from the interpretation in for instance REINFORCE or eNAC, where the policy is stochastic because the algorithm adds perturbations *itself* to foster exploration. The aim in SOC is to find the commands that minimize cost *despite* motor stochasticity that arises in the *system*, whereas in RL methods like REINFORCE and eNAC learn these commands *because* of stochasticity that these algorithms introduce *themselves*.

For the finite horizon problem, the goal is to determine the control inputs $\mathbf{u}_{t_i:t_N}$ which minimize the value function

$$V(\mathbf{x}_{t_i}) = V_{t_i} = \min_{\mathbf{u}_{t_i:t_N}} \mathrm{E}\,\boldsymbol{\tau}_i[R(\boldsymbol{\tau}_i)] \tag{10}$$

$$R(\boldsymbol{\tau}_i) = \phi_{t_N} + \int_{t_i}^{t_N} r_t\, dt \tag{11}$$

$$r_t = r(\mathbf{x}_t, \mathbf{u}_t, t) = q_t + \frac{1}{2}\mathbf{u}_t^{\intercal}\mathbf{R}\mathbf{u}_t\, dt \tag{12}$$

where $R$ is the finite horizon cost over a trajectory starting at time $t_i$ in state $\mathbf{x}_{t_i}$ and ending at time $t_N$ and where $\phi_{t_N} = \phi(x_{t_N})$ is a terminal reward at time $t_N$, $r_t = r(\mathbf{x}_t, t)$ is an arbitrary state-dependent immediate reward function, and $\mathbf{R}$ is the control cost matrix. $\boldsymbol{\tau}_i$ are trajectory pieces starting at $\mathbf{x}_{t_i}$ and ending at time $t_N$.

From SOC (Stengel, 1994), it is known that the associated Hamilton Jacobi Bellman (HJB) equation is

$$\partial_t V_t = q_t + (\nabla_{\mathbf{x}} V_t)^{\intercal}\mathbf{f}_t - \frac{1}{2}(\nabla_{\mathbf{x}} V_t)^{\intercal}\mathbf{G}_t\mathbf{R}^{-1}\mathbf{G}_t^{\intercal}(\nabla_{\mathbf{x}} V_t) + \frac{1}{2}\,\mathrm{trace}\left((\nabla_{\mathbf{xx}} V_t)\mathbf{G}_t\boldsymbol{\Sigma}\mathbf{G}_t^{\intercal}\right) \tag{13}$$

$$\mathbf{u}(\mathbf{x}_{t_i}) = \mathbf{u}_{t_i} = -\mathbf{R}^{-1}\mathbf{G}_{t_i}^{\intercal}(\nabla_{x_{t_i}} V_{t_i}) \tag{14}$$

where $\mathbf{u}(\mathbf{x}_{t_i})$ is the corresponding optimal control.

The rest of the $\mathrm{PI}^2$ derivation is dedicated to finding a solution to this non-linear, $2^{nd}$ order partial differential equation (Section 2.2.2), and applying this solution to parameterized policies (Section 2.2.3).

### 2.2.2 From Stochastic Optimal Control to Generalized Path Integral Control

Let us now briefly sketch the three main steps in the derivation of Generalized Path Integral Control (GPIC) (Theodorou et al., 2010) from the HJB equations:

1. **Linearize** the HJB into a Chapman Kolmogorov partial differential equation (PDE) by substituting $V_t = -\lambda \log \Psi_t$ (15) (Kappen, 2005) and introducing a simplification $\lambda\mathbf{R}^{-1} = \boldsymbol{\Sigma}$ (Theodorou et al., 2010).
2. Transform the Chapman Kolmogorov PDE into a **path integral** (Kappen, 2005), by using the Feynman-Kac theorem.
3. **Generalize** the path integral by partitioning the control transition matrix $\mathbf{G}_t$ into directly $\mathbf{G}_t^{(c)}$ and indirectly actuated parts (Theodorou et al., 2010).

This leads to the following path integral formulation of the value function:

$$V_t = -\lambda \log \Psi_t \tag{15}$$

$$\Psi_{t_i} = \lim_{dt \to 0} \int \exp\left(-\frac{1}{\lambda}\tilde{S}(\boldsymbol{\tau}_i)\right) d\boldsymbol{\tau}_i^{(c)} \tag{16}$$

$$\text{with } \tilde{S}(\boldsymbol{\tau}_i) = \phi_{t_N} + \sum_{j=i}^{N-1} q_{t_j} dt + \mathbf{C}_{\mathbf{t_i}} \tag{17}$$

where the integral is over paths, i.e. $d\boldsymbol{\tau}_i^{(c)} = (d\mathbf{x}_{t_i}, \ldots, d\mathbf{x}_{t_N})$. The accumulated cost $\tilde{S}(\boldsymbol{\tau}_i)$ may be interpreted as the cost-to-go from time step $i$, i.e. the cost accumulated during the rest of the trajectory starting at $t_i$. Thus at the end when $i = N$, $\tilde{S}(\boldsymbol{\tau}_i)$ only consists of the terminal cost $\tilde{S}(\boldsymbol{\tau}_N) = \phi_{t_N}$. At the beginning $i = 1$, and $\tilde{S}(\boldsymbol{\tau}_1)$ corresponds to the sum of the costs over the entire trajectory[4] . The path integral over trajectories $d\boldsymbol{\tau}_i$ in (16) may thus be interpreted as "the value at time step $i$ is (the logarithm of) the exponentiation of the cost-to-go at time step $i$ over *all* possible trajectories."

Generating *all* possible trajectories on a robot to exactly compute $V_t$ would be a time-consuming enterprise indeed. Instead, Eq. (16) may be approximated by sampling trajectories. However, in practical applications with high-dimensional state spaces, this sampling would generate primarily trajectories of high cost, and finding low cost trajectories would

[4] For completeness, the term $\mathbf{C}_{t_i}$ is $\frac{1}{2}\sum_{j=i}^{N-1}\left\|\frac{\mathbf{x}_{t_{j+1}}^{(c)}-\mathbf{x}_{t_j}^{(c)}}{dt} - \mathbf{f}_{t_j}^{(c)}\right\|^2_{\mathbf{G}_{t_j}^{(c)}\mathbf{R}^{-1}\mathbf{G}_{t_j}^{(c)\intercal}} + \frac{\lambda}{2}\sum_{j=i}^{N-1}\log\left|\mathbf{G}_{t_j}^{(c)}\mathbf{G}_{t_j}^{(c)\intercal}\right|$, cf. (Theodorou et al., 2010).

be a question of luck, rather than wisdom. Also, the dynamics of the system may bias the trajectories to be sampled in only a small part of the state space, which may not necessarily be the part where low-cost trajectories are to be found. But having a path integral that can, in principle, be estimated by performing Monte-Carlo roll-outs of the system is a big step forward from having the value function $V_t$ represented as HJB: a non-linear, second order partial differential equation which does not have a general solution (13).

Given the value function in (15), we are able to compute the optimal command. We do so by inserting the value function in (15) into the optimal command equation (14) and acquire (18). Replacing $\Psi_{t_i}$ in (18) with (16) and simplifying leads to (19).

$$\mathbf{u}_{t_i} = \lambda \mathbf{R}^{-1}\mathbf{G}_{t_i}\frac{\nabla_{x_{t_i}}\Psi_{t_i}}{\Psi_{t_i}} \tag{18}$$

$$= \int P(\boldsymbol{\tau}_i)\mathbf{D}(\boldsymbol{\tau}_i)\boldsymbol{\epsilon}(\boldsymbol{\tau}_i)d\boldsymbol{\tau}_i^{(c)} \tag{19}$$

$$\text{with } P(\boldsymbol{\tau}_i) = \frac{e^{-\frac{1}{\lambda}\tilde{S}(\boldsymbol{\tau}_i)}}{\int e^{-\frac{1}{\lambda}\tilde{S}(\boldsymbol{\tau}_i)}d\boldsymbol{\tau}_i} \tag{20}$$

$$\text{and } \mathbf{D}(\boldsymbol{\tau}_i) = \mathbf{R}^{-1}\mathbf{G}_{t_i}^{(c)\intercal}(\mathbf{G}_{t_i}^{(c)}\mathbf{R}^{-1}\mathbf{G}_{t_i}^{(c)\intercal})\mathbf{G}_{t_i}^{(c)} \tag{21}$$

Here, $P(\boldsymbol{\tau}_i)$ is the probability of trajectory $\boldsymbol{\tau}$ at time step $i$, and is inversely proportional to the cost-to-go. Lower costs thus lead to higher probabilities. The optimal command at time step $i$ is then computed as weighted average of the observed perturbation $\boldsymbol{\epsilon}(\boldsymbol{\tau}_i)$ of the trajectory, weighted by the probability of the trajectory $P(\boldsymbol{\tau}_i)$. Thus, we have inverse-cost weighted averaging, which is analogous to reward-weighted averaging as done in PoWER (8), as the inverse of a cost may be considered a reward.

As (21) makes clear, GPIC is model-based, and assumes knowledge of the system model, i.e. control matrix $\mathbf{G}(\mathbf{x}_t)$ and passive dynamics $\mathbf{f}(\mathbf{x}_t)$. In Section 2.2.3, we see that applying GPIC to parameterized policies and making the update rule iterative leads to a very powerful, model-free policy improvement algorithm.

Although GPIC does not involve a parameterized policy, we can still very loosely apply the 'fact sheet' questions from Section 2.

Fact sheet for GPIC

**Perturbation:** Action perturbation, but determined by the system. Follows directly from SOC formulation in (9). It is not to be confused with the perturbations that REINFORCE and eNAC generate themselves to foster exploration.
**Trajectory:** To perform an update, the cost and state at each time must be known. A substantial, infeasible amount of roll-outs may be required to achieve a good approximation of (16); using *all* possible paths leads to the exact solution.
**Actor-critic:** Although the value function is at the heart of GPIC's derivation, it is no longer explicitly represented, or approximated by a function approximator, in (19)-(21). Therefore, it cannot be an actor-critic.
**Update:** The concept of reward-weighted averaging is already apparent in GPIC (19), but it is not yet applied to policy parameters (GPIC does not use parameterized policies), as for instance in PoWER. To perform the update, the system model must be known.

### 2.2.3 From Generalized Path Integral Control to PI$^2$

The PI$^2$ algorithm is a special case of the GPIC optimal control solution in (19), applied to control systems with parameterized control policy (Theodorou et al., 2010) as in (22).

That is, the control command is generated from the inner product of a perturbed parameter vector $\boldsymbol{\theta} + \boldsymbol{\epsilon}_t$ with a vector of basis functions $\mathbf{g}_t$: $\mathbf{g}_t^\intercal(\boldsymbol{\theta} + \boldsymbol{\epsilon}_t)$. The noise $\boldsymbol{\epsilon}_t$ is interpreted as exploration noise sampled from a normal distribution $\mathcal{N}(0, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is a user controlled parameter. Since parameters rather than actions are perturbed, PI$^2$ is a parameter perturbing approach.

The path integral formulation in (19) applied to parameterized policies provides us with the following parameter update rule for PI$^2$:

**Policy perturbation during a roll-out**

$$\mathbf{u}_t = \mathbf{g}_t^\intercal(\boldsymbol{\theta} + \boldsymbol{\epsilon}_t) \tag{22}$$

**Parameter update for each time step through reward-weighted averaging**

$$\delta\boldsymbol{\theta}_{t_i} = \mathbf{M}_{t_i,k} \sum_{k=1}^{K} \left[P\left(\boldsymbol{\tau}_{i,k}\right)\ \boldsymbol{\epsilon}_{t_i,k}\right] \tag{23}$$

$$\text{with } P\left(\boldsymbol{\tau}_{i,k}\right) = \frac{e^{-\frac{1}{\lambda}S(\boldsymbol{\tau}_{i,k})}}{\sum_{k=1}^{K}[e^{-\frac{1}{\lambda}S(\boldsymbol{\tau}_{i,k})}]} \tag{24}$$

$$\text{and } S(\boldsymbol{\tau}_{i,k}) = \phi_{t_N,k} + \sum_{j=i}^{N-1} q_{t_j,k} + \frac{1}{2}\sum_{j=i+1}^{N-1}(\boldsymbol{\theta} + \mathbf{M}_{t_j,k}\boldsymbol{\epsilon}_{t_j,k})^\intercal\mathbf{R}(\boldsymbol{\theta} + \mathbf{M}_{t_j,k}\boldsymbol{\epsilon}_{t_j,k}) \tag{25}$$

$$\text{and } \mathbf{M}_{t_j,k} = \frac{\mathbf{R}^{-1}\mathbf{g}_{t_j,k}\ \mathbf{g}_{t_j,k}^T}{\mathbf{g}_{t_j,k}^T\mathbf{R}^{-1}\mathbf{g}_{t_j,k}} \tag{26}$$

**Weighted average over time steps**

$$[\delta\boldsymbol{\theta}]_j = \frac{\sum_{i=0}^{N-1}(N-i)\ w_{j,t_i}\ [\delta\boldsymbol{\theta}_{t_i}]_j}{\sum_{i=0}^{N-1} w_{j,t_i}(N-i)} \tag{27}$$

**Actual parameter update**

$$\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta} + \delta\boldsymbol{\theta} \tag{28}$$

The cost-to-go $S(\boldsymbol{\tau}_{i,k})$ is computed for each of the $K$ roll-outs and each time step $i = 1 \ldots N$. The terminal cost $\phi_{t_N}$, immediate costs $q_{t_i}$ and command cost matrix $\mathbf{R}$ are task-dependent and provided by the user. $\mathbf{M}_{t_j,k}$ is a projection matrix onto the range space of $\mathbf{g}_{t_j}$ under the metric $\mathbf{R}^{-1}$, cf. (Theodorou et al., 2010). The probability of a roll-out $P\left(\boldsymbol{\tau}_{i,k}\right)$ is computed as the normalized exponentiation of the cost-to-go. This assigns high probabilities to low-cost roll-outs and vice versa. The intuition behind this step is that trajectories of lower cost should have higher probabilities. The interpretation of $P_k$ as a probability follows from applying the Feynman-Kac theorem to the SOC problem, cf. (Theodorou et al., 2010). The key algorithmic step is in (23), where the parameter update $\delta\boldsymbol{\theta}$ is computed for each time step $i$ through probability weighted averaging over the exploration $\boldsymbol{\epsilon}$ of all $K$ trials. Trajectories with higher probability, and thus lower cost, therefore contribute more to the parameter update.

A different parameter update $\delta\boldsymbol{\theta}_{t_i}$ is computed for each time step. To acquire one parameter vector $\boldsymbol{\theta}$, the time-dependent updates must be averaged over time, one might simply use the mean parameter vector over all time steps: $\delta\boldsymbol{\theta} = \frac{1}{N}\sum_{i=1}^{N}\delta\boldsymbol{\theta}_{t_i}$. Although temporal averaging is necessary, the particular weighting scheme used in temporal averaging does not follow from the derivation. Rather than a simple mean, Theodorou et al. (2010) suggest the weighting scheme in Eq. (27). It emphasizes updates earlier in the trajectory, and also makes use of the activation of the $j^{\text{th}}$ basis function at time step $i$, i.e. $w_{j,t_i}$ (32).

Apart from being applied to parameterized policies rather than determining optimal controls, the key differences between GPIC and PI$^2$ are:

**PI$^2$ is model-free.** In SOC, the passive dynamics $\mathbf{f}(\mathbf{x}_t)$ and control matrix $\mathbf{G}(\mathbf{x}_t)$ represent models of the system. In the parameterized policies on which PI$^2$ acts, these are replaced with the linear spring-damper system $f_t$ and the basis functions $\mathbf{g}_t$. These are not models of the control system, i.e. the gravity vector or the inertia matrix, but rather functions that can be parameterized freely by the user.

**Perturbations are generated by PI$^2$, not the system.** Whereas in GPIC perturbations arise from the stochasticity in the system, PI$^2$ rather samples perturbations itself to actively foster exploration. These perturbation are sampled from a Gaussian $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is an open parameter set by the user. Because PI$^2$ perturbs the policy parameters ($\boldsymbol{\theta} + \boldsymbol{\epsilon}_t$), PI$^2$ is a parameter perturbing method.

**PI$^2$ is iterative.** The path integral (19) is not iterative, and computes the optimal controls from a large batch of trajectories in one go. In high-dimensional systems, many of these trajectories will be 'useless' trajectories with high cost, which do not contribute because they are assigned a low probability. To have a sufficient amount of useful trajectories, an often infeasible amount of trajectories must be sampled. PI$^2$ addresses this by applying an iterative strategy, which starts with an initial estimate of the optimal parameters $\boldsymbol{\theta}^{\text{init}}$. In robotics, this estimate is typically acquired through supervised imitation learning. PI$^2$ then samples $K$ perturbations *locally* around $\boldsymbol{\theta}^{\text{init}}$. Because $\boldsymbol{\theta}^{\text{init}}$ is assumed to already be a relatively good parameterization, local samples around $\boldsymbol{\theta}^{\text{init}}$ will in general also not be too bad. The update is therefore based on the variance within a set of trajectories that are all quite useful, rather than containing many useless trajectories. This local strategy is also applied to all subsequent parameters arising from updates. The 'single shot' global sampling from (19) has thus been replaced by a local iterative approach that searches incrementally amongst mainly good roll-outs.

As demonstrated in (Theodorou et al., 2010), PI$^2$ is able to outperform the previous RL algorithms for parameterized policy learning described in Section 2.1 by at least one order of magnitude in learning speed (number of roll-outs to converge) and also lower final cost performance. As an additional benefit, PI$^2$ has no open algorithmic parameters, except for the magnitude of the exploration noise $\boldsymbol{\Sigma}$, and the number of trials per update $K$.

Although applying our four 'fact sheet' questions to GPIC was quite forced — it does not use a parameterized policy — we may readily construct one for PI$^2$.

Fact sheet for PI$^2$

**Perturbation:** The parameters of the policy are perturbed: $\boldsymbol{\theta} + \boldsymbol{\epsilon}_t$.
**Trajectory:** To compute the cost-to-go (25), the trajectory must contain the reward received at each time step $r_{t_{i=1...N}}$. The states and actions are not needed to perform an update.
**Actor-critic:** No value function is approximated, so PI$^2$ is a direct policy search method.
**Update:** The update is based on reward-weighted averaging.

**Application to Dynamic Movement Primitives** So far, the most impressive results of PI$^2$ have been demonstrated when using Dynamic Movement Primitives (DMPs) (Ijspeert et al., 2002) as the underlying parameterized control policy. DMPs consist of a set of dynamical system equations:

$$\frac{1}{\tau}\ddot{x}_t = f_t + \mathbf{g}_t^{\intercal}\boldsymbol{\theta} \qquad \text{Transform. system} \qquad (29)$$

$$f_t = \alpha(\beta(g - x_t) - \dot{x}_t) \qquad \text{Spring-damper} \qquad (30)$$

$$[\mathbf{g}_t]_j = \frac{w_j(s_t) \cdot s_t}{\sum_{k=1}^{p} w_k(s_t)}(g - x_0) \qquad \text{Basis functions} \qquad (31)$$

$$w_j = \exp\left(-0.5 h_j (s_t - c_j)^2\right) \qquad \text{Gaussian kernel} \qquad (32)$$

$$\frac{1}{\tau}\dot{s}_t = -\alpha s_t \qquad \text{Canonical. system} \qquad (33)$$

The core idea behind DMPs is to perturb a simple linear spring-damper system $f_t$ with a non-linear component $\mathbf{g}_t^\intercal \boldsymbol{\theta}$ to acquire smooth movements of arbitrary shape. In the context of PI$^2$, the commands $\mathbf{u}_t = \mathbf{g}_t^\intercal(\boldsymbol{\theta} + \boldsymbol{\epsilon}_t)$ are thus taken to be the output of the non-linear system in (29).

The intuition of this approach is to create desired trajectories $x_{d,t}, \dot{x}_{d,t}, \ddot{x}_{d,t}$ for a motor task out of the time evolution of a nonlinear attractor system, where the goal $g$ is a point attractor and $x_0$ the start state. The (policy) parameters $\boldsymbol{\theta}$ determine the shape of the attractor landscape within a nonlinear function approximator, which allows to represent almost arbitrary smooth trajectories, e.g., a tennis swing, a reaching movement, or a complex dance movement. The canonical system $s_t$ is the phase of the movement, which is 1 at the beginning, and decays to 0 over time. The multiplication of $\mathbf{g}_t$ with $s_t$ in (31) ensures that the effect of $\mathbf{g}_t^\intercal \boldsymbol{\theta}$ disappears at the end of the movement when $s = 0$. The entire system thus converges to the goal $g$.

### 2.3 Policy Improvement through Black-box Optimization

Policy improvement may also be achieved with BBO. In general, the aim of BBO is to find the solution $\mathbf{x}^* \in X$ that optimizes the objective function $J : X \mapsto \mathbb{R}$ (Arnold et al., 2011). As in Stochastic Optimal Control and PI$^2$, $J$ is usually chosen to be a cost function, such that optimization corresponds to minimization. Let us highlight three aspects that define BBO: **1) Input**: no assumptions are made about the search space $X$; **2) Objective function**: the function $J$ is treated as a 'black box', i.e. no assumptions are made about, for instance, its differentiability or continuity; **3) Output**: the objective function returns only one scalar value. A desirable property of BBO algorithms that are applicable to problems with these conditions is that they are able to find $\mathbf{x}^*$ using as few samples from the objective function $J(\mathbf{x})$ as possible. Many BBO algorithms, such as CEM (Rubinstein and Kroese, 2004), CMA-ES (Hansen and Ostermeier, 2001) and NES (Wierstra et al., 2008), use an iterative strategy, where the current $x$ is perturbed $\mathbf{x}+\boldsymbol{\epsilon}_{k=1\dots K}$, and a new solution $\mathbf{x}^{\text{new}}$ is computed given the evaluations $J_{k=1\dots K} = J(\mathbf{x} + \boldsymbol{\epsilon}_{k=1\dots K})$.

BBO is applicable to policy improvement (Rückstiess et al., 2010b) as follows: **1) Input**: the input $x$ is interpreted as being the policy parameter vector $\boldsymbol{\theta}$. Whereas RL algorithms are tailored to leveraging the problem structure to update the policy parameters $\boldsymbol{\theta}$, BBO algorithms used for policy improvement are completely agnostic about what the parameters $\boldsymbol{\theta}$ represent; $\boldsymbol{\theta}$ might represent the policy parameters for a motion primitive to grasp an object, or simply the 2-D search space to find the minimum of a quadratic function. **2) Objective function**: the function $f$ executes the policy $\pi$ with parameters $\boldsymbol{\theta}$, and records the rewards $r_{t_{i=1:N}}$. **3) Output**: $J$ must sum over these rewards after a policy execution: $R = \sum_{i=1}^{N} r_{t_i}$ to achieve an output of only one scalar value. Examples of applying BBO to policy improvement include (Ng and Jordan, 2000; Busoniu et al., 2011; Heidrich-Meisner and Igel, 2008a; Rückstiess et al., 2010b; Marin and Sigaud, 2012; Fix and Geist, 2012).

Given the definition of BBO, we see that applying it to policy improvement already allows us categorize it in terms of the four questions in the fact sheet, without considering specific algorithms.

**Perturbation:** Since $J$ takes the policy parameters as an argument, the perturbation must also take place in this space. Therefore, all BBO approaches to policy improvement must be policy parameter perturbing methods. Furthermore, since the policy is executed

*within* the function $J$, the parameter perturbation $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ can be passed only once as an argument to $J$ *before* the policy is executed. The perturbations $\boldsymbol{\epsilon}$ therefore cannot vary over time, as is the case in REINFORCE/eNAC/PoWER; this difference is apparent when comparing the left and right illustrations in Figure 5. As Heidrich-Meisner and Igel (2008a) note: "*in evolutionary strategies [BBO] there is only one initial stochastic variation per episode, while the stochastic policy introduces perturbations in every step of the episode.*".

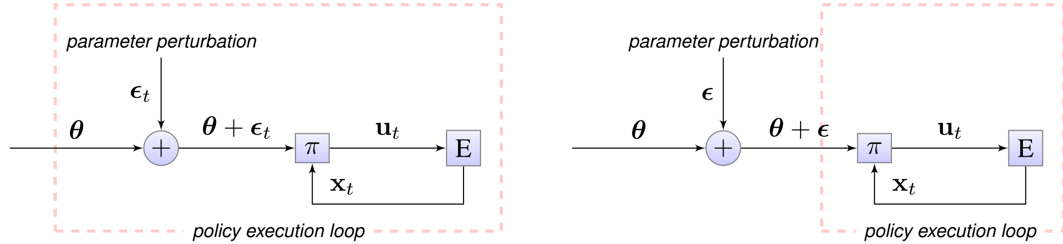


Figure 5: Left: Parameter perturbation at each time step, inside the policy execution loop (repeated from Figure 4). Right: Since policy parameters are perturbed outside the policy execution loop in BBO, they cannot vary over time.

**Trajectory:** Since the cost function returns only the total cost $R = \sum_{i=1}^{N} r_{t_i}$, it is a rather minimalist degenerate trajectory, representing the cost-to-go for $t_{i=1}$ only. Because states and actions are not stored in this 'trajectory', BBO by definition cannot make use of state/action information, as visualized in Figure 6.

This is a further reason why BBO must be parameter perturbing; using a stochastic policy *inside* the cost function $J$ would lead to $J$ returning different values for the same $\boldsymbol{\theta}$. If no information about the states visited/actions performed is available, no algorithm is able to map these differences in cost to differences in policy parameters. A defining feature of RL approaches is that they leverage information about the states that were visited, and about which states yielded which rewards (Togelius et al., 2009).

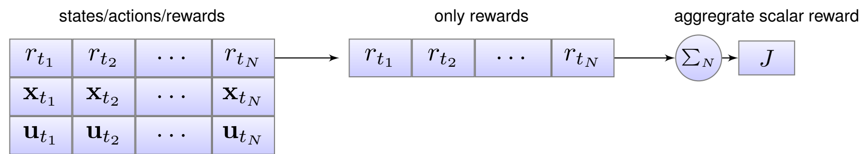


Figure 6: Illustration of the different types of information that may be stored in the trajectories that arise from policy roll-outs. Algorithms that use only the aggregate scalar reward are considered to be BBO approaches to policy improvement.

### Generic fact sheet for BBO

**Perturbation:** The parameters of the policy are perturbed ($\boldsymbol{\theta}$ +$\epsilon$), and must be constant during the execution of the policy.
**Trajectory:** A degenerate trajectory consisting of only one scalar value, representing the total cost of the trajectory: $\sum_{i=1}^{N} r_{t_i}$.
**Actor-Critic:** Since no information about states is available in the trajectories, and value functions are defined over states, BBO methods cannot approximate a value function. Therefore, they cannot be actor-critic.
**Update:** The update method depends on the algorithm. Vanilla gradients (gradient descent), natural gradients (NES) and reward-weighted averaging (CMA-ES,CEM) are all used.

In fact, *if* a policy improvement algorithm 1) uses policy perturbation, where the perturbation is constant during policy execution, *and* 2) stores only the scalar total cost of a roll-out in the trajectory, *then* it is by definition a BBO approach to policy improvement,

because the algorithm that works under these conditions is by definition a BBO algorithm. As is pointed out by Rückstiess et al. (2010b): *"the return of a whole RL episode can be interpreted as a single fitness evaluation. In this case, parameter-based exploration in RL is equivalent to black-box optimization."*. However, this distinction is not always made so clearly in the literature, where algorithms are referred to as RL or BBO based rather on their derivation and community that is being addressed. What also makes this distinction less clear is that RL is sometimes also considered to be a problem definition, rather than a solution or algorithm in itself; some of these solutions are considered to be 'typical' RL solutions to RL problems, whereas others are considered to be BBO solutions to RL problems. Where the line is drawn is a topic of vigorous debate.

We use the term 'policy improvement' for the general problem of optimizing policy parameters with respect to a utility function. BBO approaches to policy improvement are defined by the two properties above. All other approaches are RL. If a particular algorithm is preferably considered to be a RL approach, we are agnostic about this. But if the two conditions above (one scalar return, constant parameter perturbation) hold, it must at least be acknowledged that the algorithm may *also* be interpreted as being a BBO algorithm. Within the scope of this article, we define BBO and RL algorithms to be mutually exclusive; we do so for clarity only, and not to take sides in the debate.

The relative advantages and disadvantages of using the states and rewards encountered during a roll-out rather than treating policy improvement as BBO depend on the domain and problem structure, and are not yet well understood (Heidrich-Meisner and Igel, 2008a). Togelius et al. (2009, Section 4.1) list five conjectures about which approach outperforms the other for which types of RL problems.

### 2.3.1 Finite-difference methods

We now briefly present finite-difference (FD) methods, as they are one of the oldest and perhaps simplest policy gradient approaches, and may be interpreted as performing BBO on policy parameters. Here, policy parameters $\boldsymbol{\theta}$ are varied $K$ times with perturbations $\boldsymbol{\epsilon}_k$, and a regression of $\boldsymbol{\epsilon}_k$ on the resulting performance differences $\delta J_k$ is performed:

$$
\begin{aligned}
J_{\text{ref}} &= J(\boldsymbol{\theta}) && \text{Reference} \quad (34)\\
J_k &= J(\boldsymbol{\theta} + \boldsymbol{\epsilon}_k) \text{ with } k = 1 \dots K && \text{Perturb} \quad (35)\\
\delta J_{k=1:K} &= J_{k=1:K} - J_{\text{ref}} && \text{Difference} \quad (36)\\
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \left(\boldsymbol{\Delta\Theta}^{\intercal}\boldsymbol{\Delta\Theta}\right)^{-1} \boldsymbol{\Delta\Theta}^{\intercal}\boldsymbol{\Delta}\mathbf{J} && \text{Gradient} \quad (37)\\
&\quad \text{with } \boldsymbol{\Delta\Theta} = [\boldsymbol{\epsilon}_1, \dots, \boldsymbol{\epsilon}_K]^{\intercal} &&\\
&\quad \text{and } \boldsymbol{\Delta}\mathbf{J} = [\delta J_1, \dots, \delta J_K]^{\intercal} &&\\
\boldsymbol{\theta}^{\text{new}} &= \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) && \text{Update} \quad (38)
\end{aligned}
$$

This algorithm performs BBO, because $\boldsymbol{\theta}$ is passed to $J$, and $J$ is a black-box objective function that returns a scalar value. In FD, this value is a reward, rather than a cost. Note that in none of the equations above we see a policy, states, or actions; all this information is dealt with *within* the objective function $J$. The equations above may in principle also be used to find the minimum of a quadratic function. For FD, we see that $K + 1$ evaluations of the black-box objective function $J$, corresponding to $K + 1$ executions of the policy, are required to perform an update of $\boldsymbol{\theta}$. The PEGASUS algorithm (Ng and Jordan, 2000) is an example of applying the concept of finite-differencing to policy improvement.

### 2.3.2 Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES)

CMA-ES (Hansen and Ostermeier, 2001) is an example of an existing BBO method that was applied only much later to the specific domain of policy improvement (Heidrich-Meisner and Igel, 2008a). In BBO, CMA-ES is considered to be a *de facto* standard (Rückstiess et al., 2010b). We describe it a bit more extensively here, because we use CMA-ES for a comparison in Section 4.2.

CMA-ES searches for the global minimum as listed in Algorithm 1. First, CMA-ES samples $K$ exploration vectors $\boldsymbol{\epsilon}_{k=1...K}$ from a Gaussian distribution $\mathcal{N}(0, \sigma^2\Sigma)$ (line 5), where $\sigma$ is the 'step-size', and $\Sigma$ is the covariance matrix. The cost $J_k$ of each of these samples is then computed with the black-box cost function $J$ (line 7). The exploration vectors are then sorted with respect to their cost, and only the best $K_e$ samples are kept (line 9). Each of these $K_e$ 'elite' samples is assigned a weight. The function that maps the cost $J_k$ to the weight $P_k$ are chosen by the user, and must satisfy the constraints $\sum_{k=1}^{K_e} P_k = 1$ and $P_1 \geq \cdots \geq P_{K_e}$. Thus, samples with lower cost are assigned larger weights than those with higher cost. The default suggested by Hansen and Ostermeier (2001) is listed in line 11, i.e. $P_k = ln\left(\max\left(K/2, K_e\right) + 0.5\right)\right) - ln(k)$. The parameter update is then computed with

```
1  while cost not converged do
2      Exploration
3      foreach k in K do
4          Sample exploration vector from Gaussian
5          ε_k ~ N(0, σ²Σ) ;
6          Determine cost of the sample
7          J_k = J(θ + ε_k)
8      Compute weights (probabilities)
9      ε_{k=1...K} ← sort ε_{k=1...K} w.r.t J_{k=1...K}
10     foreach k in K do
11         P_k = { ln(max(K/2, K_e) + 0.5)) − ln(k)   if k ≤ K_e
                 { 0                                   if k > K_e
12     Update mean: Reward-weighted averaging over K trials
13     δθ = Σ_{k=1}^{K} [P_k ε_k]
14     Update covariance matrix: Reward-weighted averaging
15     Σ^tmp = Σ_{k=1}^{K_e} P_k ε_k ε_k^T
16     Update parameters
17     θ ← θ + δθ
18     Update Covariance Matrix using Evolution Paths
19     p_σ ← (1 − c_σ) p_σ + sqrt(c_σ(2 − c_σ)μ_P) Σ^{−1} (δθ/σ)
20     σ_new = σ × exp( (c_σ/d_σ) ( ||p_σ|| / E||N(0,I)|| − 1 ) )
21     p_Σ ← (1 − c_Σ) p_Σ + h_σ sqrt(c_Σ(2 − c_Σ)μ_P) (δθ/σ)
22     Σ^new = (1 − c_1 − c_μ) Σ + c_1(p_Σ p_Σ^T + δ(h_σ)Σ) + c_μ Σ^tmp
```

**Algorithm 1**: The CMA-ES algorithm.

reward-weighted averaging using the weights $P_k$ (line 13), i.e. $\delta\boldsymbol{\theta} = \sum_{k=1}^{K} [P_k \boldsymbol{\epsilon}_k]$. The covariance matrix is also updated using reward-weighted averaging (line 15).

The last part (lines 19-22) further adapts the step-size $\sigma$ and covariance matrix $\Sigma$, using the so-called 'evolution paths' $p_\sigma$ and $p_\Sigma$, which store information about previous parameter updates. Although these last lines lead to more robust convergence in practice, we do not elaborate on them here, as they do not involve the 'core' of the algorithm, and are not relevant for our purposes. For a full explanation of the algorithm we refer to Hansen and Ostermeier (2001).

### 2.3.3 Other BBO Algorithms Applied to Policy Improvement

Further BBO algorithms that have been applied to policy improvement include the Cross-Entropy Method (CEM) (Rubinstein and Kroese, 2004; Busoniu et al., 2011; Heidrich-Meisner and Igel, 2008b), which is very similar to CMA-ES, but has simpler methods for determining the weights $P_k$ and performing the covariance matrix update. For a comparison of further BBO algorithms such as PGPE (Rückstiess et al., 2010b) and Natural Evolution Strategies (NES) (Wierstra et al., 2008) with CMA-ES and eNAC, please see the overview article by Rückstiess et al. (2010b).

NEAT+Q is a hybrid algorithm that actually *combines* RL and BBO in a very original way (Whiteson and Stone, 2006). Within our classification, NEAT+Q is first and foremost an actor-critic approach, as function approximation is used to explicitly represent the value function. What sets NEAT+Q apart is that the representation of the value function is evolved through BBO, with the "Neuro Evolution of Augmenting Topologies" (NEAT). This alleviates the user from having to design this representation; unsuitable representations may keep RL algorithms from being able to learn the problem, even for algorithms with proven convergence properties (Whiteson and Stone, 2006).

## 2.4 Classification of Policy Improvement Algorithms

Figure 7 is an extended version of Figure 1, which now includes all the questions of our fact sheet. This table will be especially useful for our discussion in Section 5 about which classes of algorithms are appropriate for which classes of problems. The figure also reminds us of the main aim of this article: continuing the SOC →GPIC →PI$^2$ stream to acquire the PI$^{BB}$ algorithm.

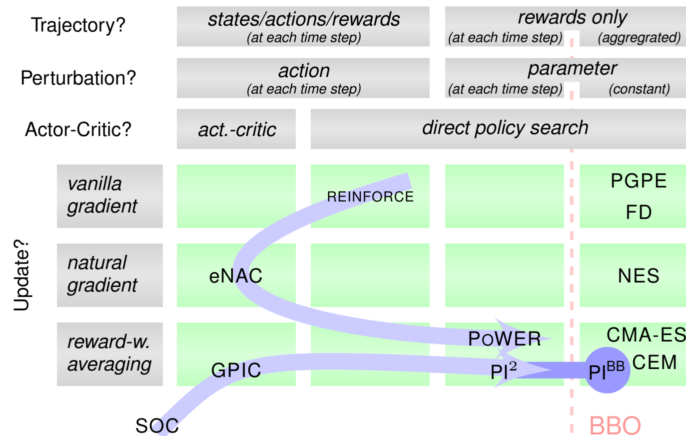


Figure 7: Classification of policy improvement algorithms, given their fact sheets. The two arrows represent chronology and order of derivation.

# 3 From PI$^2$ to PI$^{BB}$

Figure 8 repeats the 'stream' from SOC to GPIC to PI$^2$. In terms of exploration and parameter updates, it becomes apparent that PI$^2$ is similar to the BBO algorithm CMA-ES. In fact, they are so similar that a line-by-line comparison of the algorithms is feasible, as is done by Stulp and Sigaud (2012).

Figure 8 suggests that two modifications are required to convert PI$^2$ into a BBO algorithm. First of all, the policy perturbation method must be adapted. PI$^2$ and BBO both use policy parameter perturbation, but in PI$^2$ it varies over time ($\boldsymbol{\theta} + \boldsymbol{\epsilon}_t$), whereas it must remain

constant over time in BBO ($\boldsymbol{\theta} + \boldsymbol{\epsilon}$). In Section 3.1, we therefore simplify the perturbation method in PI$^2$ to be constant over time, which yields the algorithm variation PI$^{2-}$.

Second, we must adapt PI$^2$ such that it is able to update the parameters based only on the scalar aggregated cost $J = \sum_{i=1}^{N} r_{t_i}$, rather than having access to the reward $r_t$ at each time step $t$. This is done in Section 3.2, and yields the PI$^{\text{BB}}$ algorithm. An important aspect of these two simplifications for deriving PI$^{\text{BB}}$ from PI$^2$ is that they do not violate any of the assumptions required for the derivation of PI$^2$ from SOC, which we motivate in more detail throughout this section.

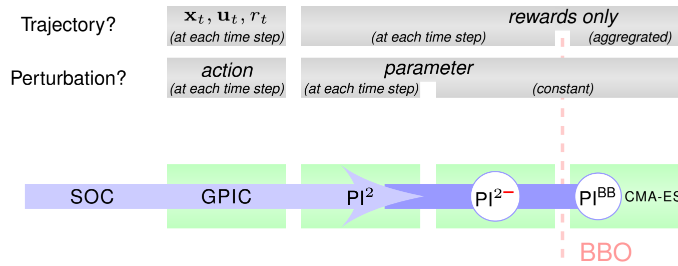


Figure 8: Simplifications to PI$^2$ to derive PI$^{\text{BB}}$, a BBO algorithm.

### 3.1 Simplifying the Exploration Method

This section is concerned with analyzing exploration in PI$^2$. In particular we: 1) Argue on theoretical grounds why exploration vectors $\boldsymbol{\epsilon}$ must not be sampled anew at each time step during a roll-out, and present two previously proposed alternative exploration methods (Theodorou et al., 2010; Tamosiumaite et al., 2011; Stulp and Sigaud, 2012). These exploration variants of PI$^2$ are denoted PI$^{2\sim}$/PI$^{2\sqcap}$/PI$^{2-}$. 2) Show that these three methods lead to different levels of exploration in policy space, and that compensating for this effect leads the methods to perform essentially identical.

Especially the last point provides a deeper insight into the underlying cause for the performance differences that have been observed for the three exploration methods, and is one of the contributions of this article. Furthermore, demonstrating that constant exploration (PI$^{2-}$) is not outperformed by the other two exploration methods (PI$^{2\sim}$/PI$^{2-}$) paves the way for our comparison between episodic RL and BBO in Section 4.1.

#### 3.1.1 Parameter Perturbation in the Context of the PI$^2$ Derivation

As discussed in Section 2.2.3, applying GPIC to parameterized policies, which yields PI$^2$, has several consequences:

- In GPIC, actions are perturbed, whereas PI$^2$ is a parameter perturbing method. This represents the left-to-right stream in Figure 8.
- Perturbations are no longer caused by the system, but rather generated by the PI$^2$. Therefore, $\boldsymbol{\Sigma}$ is an open parameter, rather than determined by the system. In PI$^2$, $\boldsymbol{\Sigma} = \lambda \mathbf{R}^{-1}$, where the $\lambda$ is the parameter that controls the magnitude of exploration, and $\mathbf{R}$ is the command cost matrix. Thus, the scalar $\lambda$ determines the magnitude of the exploration.
- In SOC, it is assumed that the stochasticity is independent of time, and $\boldsymbol{\epsilon}_{t_i}$ is therefore different for each time step $t_i$. If $\boldsymbol{\epsilon}_{t_i}$ would be constant over time, it would be a bias (e.g. a 1 degree offset due to a calibration error in a robot joint) rather than noise (e.g. stochasticity arising from noisy encoders). The PI$^2$ algorithm inherits this property

through its derivation from SOC, and thus also has time-varying perturbations $\boldsymbol{\epsilon}_{t_i}$. Note a subtle difference between the two: in SOC a time-varying perturbation $\boldsymbol{\epsilon}_t$ is added to a time-varying command $\mathbf{u}_t$, whereas in PI$^2$ a time-varying perturbation $\boldsymbol{\epsilon}_t$ is added to a *constant* parameter vector $\boldsymbol{\theta}$.

An important result of this last point is that the stochasticity must, in principle, no longer be time-independent, as is the case when applying it to motor commands. In practice, time-varying exploration has several disadvantages, which have been pointed out in Section 2.1.3. In practical applications of PI$^2$, the noise is therefore not varied at every time step (Theodorou et al., 2010; Tamosiumaite et al., 2011; Stulp et al., 2012).

### 3.1.2 Three Proposed Methods for Parameter Perturbation

We refer to the ‘canonical’ version of PI$^2$, which samples different exploration vectors $\boldsymbol{\epsilon}_t$ for each time step, as PI$^{2\sim}$. The small blue symbol serves as a mnemonic to indicate that exploration varies at a high frequency, as seen in the upper left graph of Figure 9, which plot $\boldsymbol{\epsilon}_t$ against time $t$.

As an alternative to time-varying exploration, Theodorou et al. (2010) propose to generate exploration noise only for the basis function with the highest activation. We refer to this second method as PI$^2$ with exploration per basis function, or PI$^{2\sqcap}$, where the green graphs serves as a mnemonic of the shape of the exploration for one basis function. The difference between PI$^{2\sim}$ and PI$^{2\sqcap}$ is visualized in the top row of Figure 9, which depicts $\boldsymbol{\epsilon}_t$ over time for an exploration magnitude[5] of $\lambda = 0.05$.

Alternatively, $\boldsymbol{\epsilon}_{t_i,k}$ can be set to have a constant value during a roll-out. Thus, for each of the $K$ roll-outs, we generate $\boldsymbol{\epsilon}_k$ exploration vectors before executing the policy, and keep it constant during the execution, i.e. $\boldsymbol{\epsilon}_{t_i,k} = \boldsymbol{\epsilon}_k$. We call this ‘PI$^2$ with constant exploration’, and denote it as PI$^{2-}$, where the horizontal line indicates a constant value over time. Note that $\boldsymbol{\epsilon}_k$ will still have a temporally extended effect, because it is multiplied with a basis function that is active throughout an extended part of the movement, as depicted in the right graph, second row in Figure 9.

**Exploration in Parameter Space vs. Exploration in Policy Output Space** The third row of Figure 9 depicts the cumulative activation of the (third) perturbed basis function: $\sum_{s=0}^{t} \mathbf{g}_t \boldsymbol{\epsilon}_t$. For PI$^{2\sim}$ (left), we see that, since consecutive positive/negative perturbations cancel each other out, the cumulative activation is quite low. For PI$^{2\sqcap}$ (center), where there is no canceling out, the cumulative activation is much higher. For PI$^{2-}$ (right) it is the highest, because the exploration vector $\boldsymbol{\epsilon}_t$ is never 0, and thus has the largest effect when multiplied with $\mathbf{g}_t$.

Since $\mathbf{g}_t \boldsymbol{\epsilon}_t$ directly determines the acceleration of the DMP output (29), higher cumulative activations lead to higher accelerations. This becomes apparent in the fourth row of Figure 9, which depicts 50 roll-outs of the DMP, all sampled with an exploration magnitude of $\lambda = 0.05$. Upon visual inspection of the trajectories in the fourth row of Figure 9, the variance for PI$^{2-}$ is higher than for the other two. To quantify this effect, we perform $K = 1000$ roll-outs, and determine the standard deviation of the DMP output at time $t$ as: $\sigma_t^x = \sqrt{\frac{1}{K-1} \sum_{k=1}^{K} \left( x_t^k - \bar{x}_t \right)^2}$. The solid dark graphs in the final row in Figure 9 depict this standard deviation for the 1000 roll-outs for the three exploration methods.

If we set the exploration magnitude for PI$^{2\sim}$ to a higher value of $\lambda = 0.170$, we see that the standard deviation (dashed blue line, lower left graph) in policy output space ($\sigma_t^x$)

[5] In this section, we choose $\mathbf{R} = \mathbf{I}_B$, where B is the number of basis functions, such that $\boldsymbol{\Sigma} = \lambda \mathbf{I}^{-1} = \lambda \mathbf{I}$. This is convenient, because the magnitude of exploration in parameter space is determined solely by the scalar $\lambda$. Higher $\lambda$ thus means more exploration in parameter space, and therefore more variance in the output trajectories of the DMP.

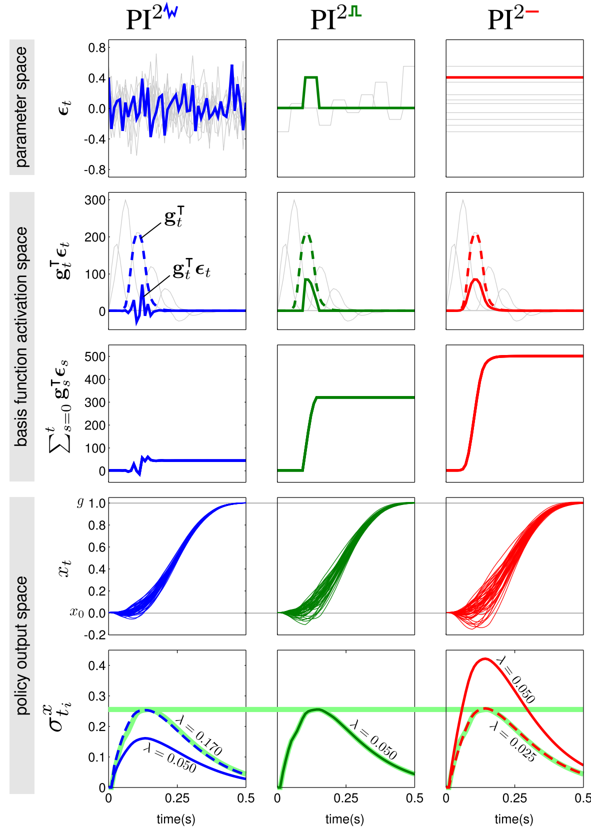


Figure 9: Visualization of the three different forms of exploration, using one roll-out of a 1-D DMP of duration 0.5s. The DMP has 10 basis functions, of which the third is highlighted. Top row: exploration vector $\epsilon_t$ over time. Second row: The 10 basis functions activations $\mathbf{g}_t$ (light grey and highlighted dashed), and the exploration vector multiplied with the basis function activation $\mathbf{g}_t^\mathsf{T}\epsilon_t$ (solid). Third row: Cumulative activation of the third (highlighted) basis functions. Fourth row: Output of the DMP $x_t$ for 50 roll-outs. Final row: Standard deviation in the policy output during the movement, averaged over 1000 roll-outs.

develops the same as PI$^{2\sqcap}$ for $\lambda = 0.05$. We use PI$^{2\sqcap}$ with $\lambda = 0.05$ as a reference, because this is the exploration method and exploration magnitude used by Theodorou et al. (2010). For PI$^{2-}$, the story is the converse, and a lower exploration magnitude of $\lambda = 0.025$ is required to achieve the same exploration magnitude in policy output space, cf. the red dashed graph.

**Summary:** By setting the policy parameter exploration magnitude $\lambda$ appropriately, we achieve essentially the same exploration in the policy output for PI$^{2\text{w}}$, PI$^{2\sqcap}$ and PI$^{2-}$. The practical implications and relation to previous work (Theodorou et al., 2010; Tamosiumaite et al., 2011; Stulp and Sigaud, 2012) of this is made clear in the empirical comparison that follows.

### 3.1.3 Empirical Comparison

The experiments in this article are based on the same tasks as presented by Theodorou et al. (2010). The main advantage of using these tasks is that it allows for a direct comparison with the results reported by Theodorou et al. (2010). The tasks are described in Appendix A.

For each learning session, we are interested in comparing the convergence speed and final cost, i.e. the value to which the learning curve converges. Convergence speed is measured as the parameter update after which the cost drops below $5\%$ of the initial cost before learning. The final cost is the mean cost over the last 100 updates. For all tasks and algorithm settings, we execute 10 learning sessions (which together we call an 'experiment'), and report the $\mu \pm \sigma$ over these 10 learning sessions. For all experiments, the DMP and PI$^2$ parameters are the same as in (Theodorou et al., 2010), and listed in Appendix A.

Figure 10 summarizes the results of comparing the different exploration methods on the example Task 2; Figure 11 presents the results for the other tasks. The top graph represents the learning curves ($\mu \pm \sigma$) over 10 learning sessions, with exploration magnitude $\lambda = 0.05$ for all exploration methods.

The left graphs enables the comparison of convergence speed. To evaluate the convergence speed, we determine when each of the learning curves drops below 5% of the cost before learning. The means of these values for the three exploration methods are visualized as vertical lines in the left graph of Figure 10. At the top of these lines, a horizontal bar represents the standard deviation. For convergence we see that PI$^{2-}$ <PI$^{2\sqcap}$ <PI$^{2\mathrm{W}}$ ($p$-value $< 0.001$), i.e. constant exploration converges quickest.

The right graphs compare the final cost of each method, but depicts the average learning curve during the last 100 updates, after which all learning curves have converged. The vertical lines and horizontal bars in the right graphs visualize the $\mu \pm \sigma$ of the final cost over the 10 learning sessions, where the final cost is defined as the mean over a learning curve during the last 100 updates. For the value and variance in the final cost, we see that PI$^{2\mathrm{W}}$ <PI$^{2\sqcap}$ <PI$^{2-}$ ($p$-value $< 0.001$), i.e. this time PI$^{2\mathrm{W}}$ performs significantly better than the other two methods.

Generating exploration only for the basis function with the highest approximation (PI$^{2\sqcap}$) thus provides a good trade-off between achieving fast convergence and a low final cost, which is why it has been independently recommended in different applications of PI$^2$ (Theodorou et al., 2010; Tamosiumaite et al., 2011). If convergence speed is the most important feature, we have argued that constant exploration is best (Stulp and Sigaud, 2012).

However, the second row of graphs shows that, when we normalize for the variance in the policy output by setting $\lambda = 0.170/0.050/0.025$ for PI$^{2\mathrm{W}}$/PI$^{2\sqcap}$/PI$^{2-}$ respectively, as discussed in Section 3.1.2, that the difference between the PI$^2$ variations do not differ signicantly ($p$-value $> 0.07$ for all pairwise comparisons) and the mean and variance in the final cost is almost identical ($p$-value $> 0.67$).

**Decaying Exploration as Learning Progresses** The value and variance in final cost is still quite high when considering the top two rows in Figure 10. A typical reason for high variance in the final cost is that the high exploration that was suitable at the beginning of learning prevents the algorithm from converging to the lowest possible cost at the end of learning. For this reason, exploration is often decayed exponentially as learning progresses (Stulp and Sigaud, 2012). The bottom graphs of Figure 10 depicts the results for the three exploration methods, with a decay factor of $\gamma = 0.98$, i.e. the exploration at update $u$ is determined by $\mathbf{\Sigma}_u = \gamma^u \lambda \mathbf{I}$. Here $\lambda = \{0.05, 0.170, 0.025\}$ as above, for normalized exploration in policy output space. Again, differences in convergence speed are not significant ($p$-value $> 0.05$), and final cost the final cost is 0 for all exploration methods. This is the minimal possible cost, corresponding to passing through the via-point perfectly.

**Results on All Tasks** Figure 11 summarizes the convergence speed and final cost for all five tasks described in the Appendix A, where the $\lambda$ has been normalized to achieve the same variance in policy output, and with decaying exploration. For each task, all values have been normalized w.r.t. the value for PI$^{2\sqcap}$. For instance, for Task 2, the convergence below 5% of the initial cost in the bottom graph of Figure 10 was on average at updates 14.7, 13.7, and 13.0 for PI$^{2\mathrm{W}}$,PI$^{2\sqcap}$, and PI$^{2-}$. Normalized for PI$^{2\sqcap}$, this becomes 1.07, 1.00 and 0.95, as highlighted in Figure 11.
From this bar plot, we derive the following conclusions:

- The final costs do not differ much between the exploration methods. On average it is 2.2% higher than for PI$^{2\sqcap}$, with a maximum of 6.6% for Task 5. The differences are only significant for Task 1 & 5 ($p$-value $< 0.05$).

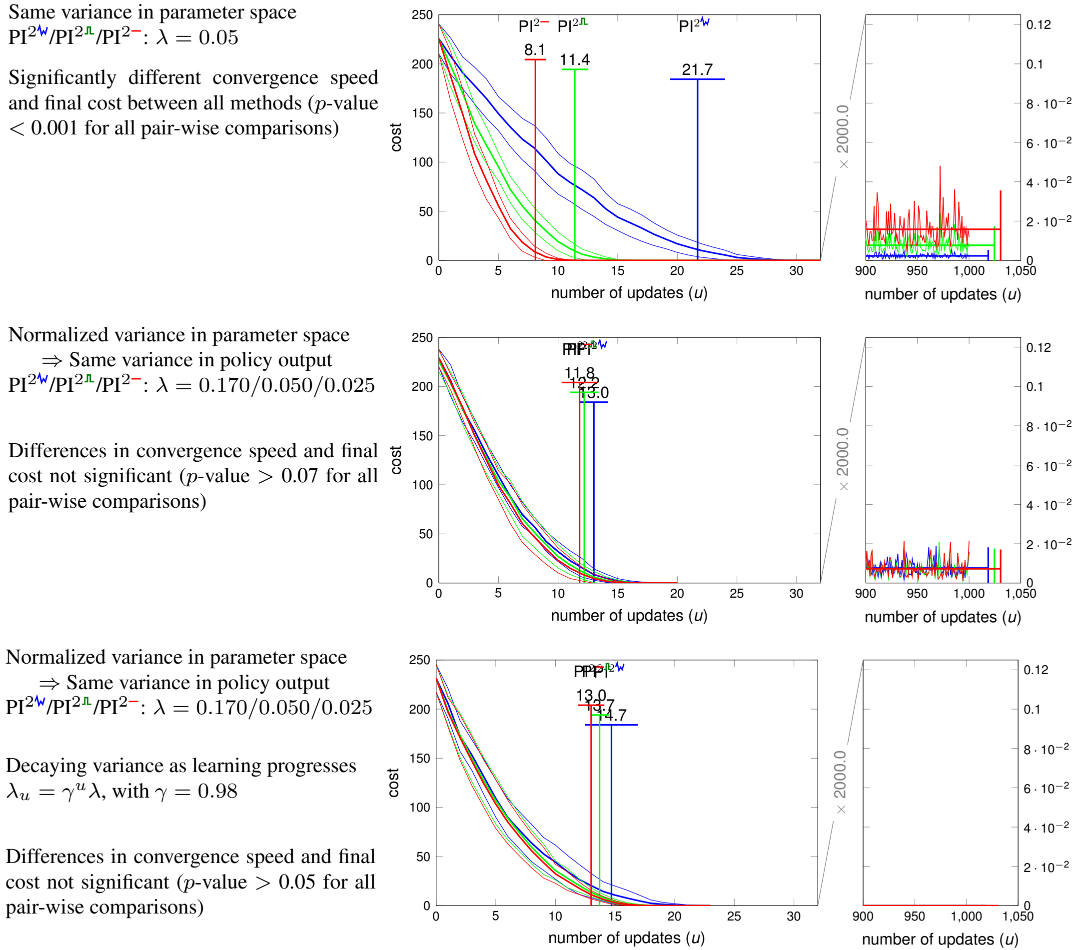


Figure 10: Learning curves for the three different methods of generating exploration ($PI^{2\mathrm{W}}$, $PI^{2\sqcap}$ and $PI^{2-}$) for Task 2. The three rows represent different parameter settings of the learning algorithm. The left graphs, which highlight differences in convergence speed, shows the learning curves ($\mu \pm \sigma$ over 10 separate learning sessions) during the first 32 updates, which corresponds to 480 trials. The right graphs highlight the cost after convergence, and depicts the learning curves (only $\mu$ for clarity) between updates 900 and 1000. The $y$-axis is zoomed $\times 2000$ in comparison to the left graph. Annotations are described in the text.

- The convergence speed varies by 10% between exploration methods, except for Task 1, where they differ by almost 20%. The convergence speed differs significantly between methods only for Task 1 ($p$-value $< 0.05$).
- Convergence speed is inversely related to final cost. That is, if a method has a faster convergence than the baseline, it will have a higher final cost. This represents the general trade-off between convergence speed and final cost. Note that this does not hold for Task 2 and 3, because the final cost goes to 0 for all methods.

### 3.1.4 Conclusion for Exploration Methods

In conclusion, the faster convergence as observed with per-basis (Theodorou et al., 2010; Tamosiumaite et al., 2011) or constant (Stulp and Sigaud, 2012) exploration noise does not seem to be caused by intrinsic properties of the exploration method, but rather by the higher level of exploration they lead to in the output of the policy. When choosing the parameter exploration magnitude such that it leads to the same amount of exploration in task space, the exploration methods have much more similar convergence speed and final cost, and which is faster or slower depends on the task. For all methods, the advantages of high initial exploration for fast convergence *and* exploitation of the learned policy after learning may be achieved by decaying exploration over time.

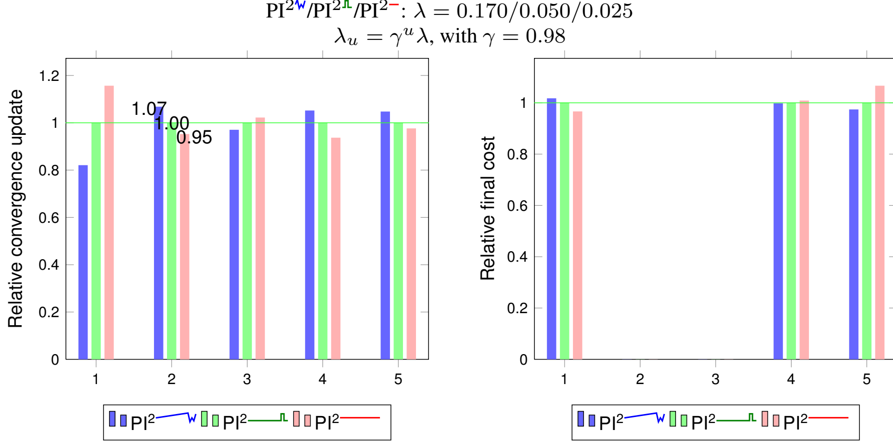


Figure 11: Summary of the results on all five experiments with normalized and decaying exploration.

### 3.2 Simplifying the Parameter Update

In this section, we simplify the parameter update rule of PI$^2$ which yields the simpler PI$^{BB}$ algorithm. We motivate why this simplication is valid within the PI$^2$ derivation, and empirically compare PI$^{2-}$ and PI$^{BB}$.

#### 3.2.1 Temporal Averaging in the Context of the PI$^2$ Derivation

In PI$^2$, a different parameter update $\delta\boldsymbol{\theta}_{t_i}$ is computed for each time step $i$. This is caused by its derivation from GPIC, where motor commands $u_t$ are different at each time step; it is difficult to imagine a task that requires a constant motor command during a roll-out. But since the policy parameters $\boldsymbol{\theta}$ are constant during a roll-out, there is a need to condense the $N$ parameters updates $\delta\boldsymbol{\theta}_{t_i=0:N}$ into one update $\delta\boldsymbol{\theta}$. This step is called temporal averaging, and was proposed by Theodorou et al. (2010) as:

$$[\delta\boldsymbol{\theta}]_d = \frac{\sum_{i=1}^{N}(N-i+1)\; w_{d,t_i}\; [\delta\boldsymbol{\theta}_{t_i}]_d}{\sum_{i=1}^{N} w_{d,t_i}(N-i+1)}. \tag{39}$$

This temporal averaging scheme emphasizes updates earlier in the trajectory, and also makes use of the basis function weights $w_{d,t_i}$. However, since this does not directly follow from the derivation *"[u]sers may develop other weighting schemes as more suitable to their needs."* (Theodorou et al., 2010). As an alternative, we now choose a weight of 1 at the first time step, and 0 for all others. This means that all updates $\delta\boldsymbol{\theta}_{t_i}$ are ignored, except the first one $\delta\boldsymbol{\theta}_{t_1}$, which is based on the cost-to-go at the first time step $S(\boldsymbol{\tau}_{1,k})$. By definition, the cost-to-go at $t_1$ represents the cost of the entire trajectory. This implies that we must only compute the cost-to-go $S(\boldsymbol{\tau}_{i,k})$ and probability $P(\boldsymbol{\tau}_{i,k})$ for $i = 1$. This simplified PI$^2$ variant, which does not use temporal averaging, and which we denote 'PI$^{BB}$' is presented in more detail in Section 4.

Note that this simplification depends strongly on using constant exploration noise during a roll-out. If the noise varies at each time step or per basis function, the variation at the first time step $\boldsymbol{\epsilon}_{t_1,k}$ is not at all representative for the variations throughout the rest of the trajectory. It is therefore more accurate to consider PI$^{BB}$ as a variant of PI$^{2-}$, rather than of PI$^2$ $\equiv$PI$^{2}$.

#### 3.2.2 Empirical Comparison

We evaluate the effect of temporal averaging by comparing PI$^{2-}$ (with constant exploration) and PI$^{BB}$ (which does not use temporal averaging and has constant exploration by default),

which is also executed in 10 learning sessions which 1000 updates each. These learning curves ($\mu\pm\sigma$) for both non-decaying ($\gamma = 1.0$) and decaying ($\gamma = 0.98$) exploration are depicted in Figure 12. We see that $\mathrm{PI}^{\mathrm{BB}}$ converges almost twice as fast as $\mathrm{PI}^{2-}$, and that both converge to a final cost of 0.

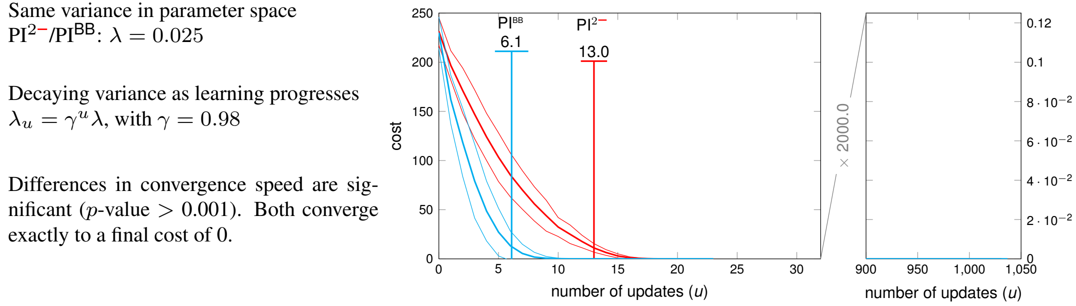


Figure 12: As Figure 10, but for $\mathrm{PI}^{2-}$ (repeated in red) and $\mathrm{PI}^{\mathrm{BB}}$ (cyan) only.

Figure 13 repeats the convergence speed and final cost from Figure 11 for all tasks, but adds the values for $\mathrm{PI}^{\mathrm{BB}}$ for comparison. This bar plot reveals the following:

- $\mathrm{PI}^{\mathrm{BB}}$ achieves a substantially faster convergence speed than the other $\mathrm{PI}^{2}$ variants methods. On average it is 53% of the convergence speed for $\mathrm{PI}^{2\sqcap}$. This is significant for all pair-wise comparisons ($p < 0.01$), except for the difference between $\mathrm{PI}^{2\mathrm{W}}$ and $\mathrm{PI}^{\mathrm{BB}}$ for Task 1 ($p > 0.894$).
- $\mathrm{PI}^{\mathrm{BB}}$ achieves equivalent or better final costs than $\mathrm{PI}^{2-}$. On average, the final cost is 6% lower than for the reference $\mathrm{PI}^{2\sqcap}$. This is significant for all pair-wise comparisons ($p < 0.002$), except for the difference between $\mathrm{PI}^{2\mathrm{W}}$ and $\mathrm{PI}^{\mathrm{BB}}$ for Task 5 ($p > 0.150$) and Task 2 and 3, where all methods converge to a final cost of 0.

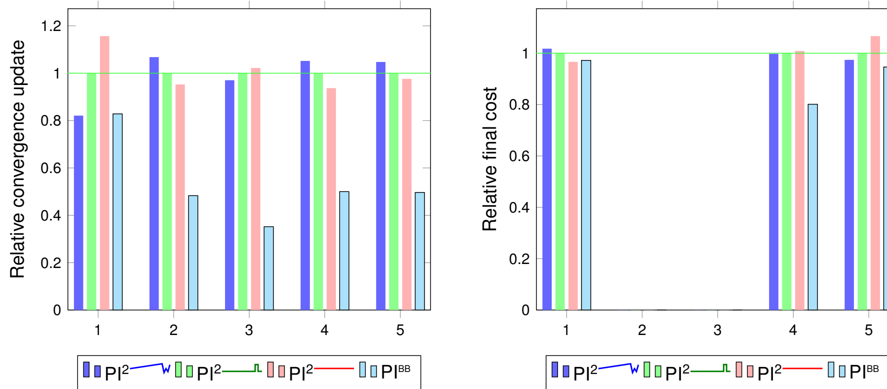


Figure 13: Summary of the results on all five experiments, including $\mathrm{PI}^{\mathrm{BB}}$.

### 3.2.3 Conclusion for Parameter Update

For the five tasks considered in this article (and thus those in (Theodorou et al., 2010), we see that $\mathrm{PI}^{\mathrm{BB}}$ achieves equal or better performance in terms of convergence speed and final cost than the other $\mathrm{PI}^{2}$ variants, if we decay exploration over time.

## 4 The $\mathrm{PI}^{\mathrm{BB}}$ Algorithm

In Section 3, we have defined a variant of $\mathrm{PI}^{2}$ called $\mathrm{PI}^{\mathrm{BB}}$, in which: 1) Exploration noise is constant over time, i.e. as in $\mathrm{PI}^{2-}$. 2) Temporal averaging uses only the first update, i.e.

$\delta\boldsymbol{\theta}^{\text{new}} = \delta\boldsymbol{\theta}^{\text{new}}{}_{t_1}$. We also refer to this simply as 'no temporal averaging'. As previously discussed in Section 3, these simplifications do not violate any of the assumptions made when deriving PI$^2$ from SOC. In this section, we perform a closer analysis of PI$^{\text{BB}}$.

Figure 14 lists both the PI$^2$ (left) and PI$^{\text{BB}}$ (center) algorithms. Since PI$^{\text{BB}}$ is a simplified version of PI$^2$, we have visualized the simplifications as dark red areas, that indicate that these lines are dropped from the PI$^2$ algorithm. In Figure 14, simplifications have been labeled: (C1) – keep exploration constant; (C2) – do not use temporal averaging; (M) – drop the projection matrix $\mathbf{M}$ from the parameter update.

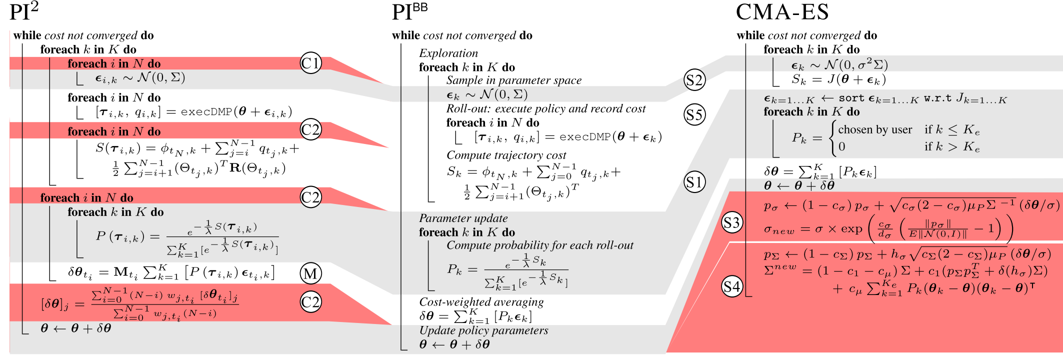


Figure 14: Comparison of PI$^2$ (left), PI$^{\text{BB}}$ (center) and CMA-ES (right)

## 4.1 PI$^{\text{BB}}$ is a Black-Box Optimization Algorithm

We now demonstrate that the PI$^{\text{BB}}$ algorithm is equivalent to applying a BBO algorithm to the policy parameters.

The effect of using constant exploration is that PI$^{\text{BB}}$ has only one remaining loop over time (to execute the policy), and that temporal averaging (the penultimate line of PI$^2$) disappears, cf. Figure 14. As a consequence, determining the cost of a vector $\boldsymbol{\theta} + \boldsymbol{\epsilon}_k$ may now be interpreted as a black-box cost function, i.e. $S_k = J(\boldsymbol{\theta} + \boldsymbol{\epsilon}_k)$ with the perturbed policy parameters (which do not vary over time due to (C1)) as input, and the scalar trajectory cost $S_k$ as output (only the entire trajectory cost is needed due to (C2)). In PI$^{\text{BB}}$, the cost function $S_k = J(\boldsymbol{\theta} + \boldsymbol{\epsilon}_k)$ thus does the following: 1) integrate and execute the policy with constant parameters $(\boldsymbol{\theta}_k + \boldsymbol{\epsilon}_k)$; 2) record the costs at each time step during the execution; 3) when the roll-out is done, sum over the costs, and return them as $S_k$.

## 4.2 PI$^{\text{BB}}$ is a special case of CMA-ES

Now, we show more specifically that PI$^{\text{BB}}$ is a special case of the CMA-ES algorithm.

We now describe several simplifications/specializations of the CMA-ES algorithm, listed in Section 2.3.2. Our intention is not to create a more efficient algorithm — we will remove some core functionality of CMA-ES in the process — but rather to highlight the relationship between PI$^{\text{BB}}$ and CMA-ES.

In CMA-ES, the weighting function may be chosen freely, as long as the conditions $\sum_{k=1}^{K_e} P_k = 1$ and $P_1 \geq \dots \geq P_{K_e}$ hold. Since the exponentiation of the cost in PI$^2$ meets these conditions, we set $K_e = K$, and the function that maps $S_k$ to $P_k$ to that of PI$^2$. This step is labeled (S1) in Figure 14.

The next step is to disable the covariance matrix adaptation in CMA-ES, which is done as follows: (S2) Set the initial step-size $\sigma = 1$ (S3) Disable step-size updating, by setting the time horizon $c_\sigma = 0$. This makes (20) collapse to $\sigma_{new} = \sigma \times \exp(0)$, which means the step-size stays equal over time. Since the initial step-size $\sigma = 1$, $\sigma$ simply drops from all equations. (S4) Disable covariance matrix updating, by setting $c_1 = 0$ and $c_\mu = 0$. The

second and third terms of (22) then drop, and what remains is $(1 - 0 - 0)\Sigma$. Therefore, the covariance matrix is not adapted, and remains constant during learning. Setting the parameters as listed above thus makes the entire covariance matrix adaptation drop[6].

Finally, CMA-ES can readily be applied to policy improvement, as is done in (Heidrich-Meisner and Igel, 2008a), by considering the cost $S_k$ to be the cost of the trajectory that arises when executing the policy, labeled (S5) in Figure 14.

Simplifications (S1)-(S5) are again visualized as red areas in Figure 14. Interestingly, the algorithm that arises from applying (S1)-(S5) to CMA-ES is *equivalent* to PI$^{BB}$. Note that CMA-ES was not modified in any way, we simply set certain open parameters of CMA-ES to specific values.

**Summary:** PI$^{BB}$, which is a variant of PI$^2$ with constant exploration and without temporal averaging, is a BBO algorithm and in particular a special case of CMA-ES without covariance matrix updating.

It is important to recognize that PI$^{BB}$ is, compared to other state-of-the-art BBO algorithms, quite simplistic. Our reasons for introducing PI$^{BB}$ are: 1) To be able to compare two algorithms (PI$^2$/PI$^{BB}$) that differ *only* in being RL or BBO methods, but are identical otherwise. 2) To demonstrate that a degenerate version of PI$^2$ (with constant exploration and without temporal averaging) is equivalent to a degenerate version of CMA-ES (without covariance matrix adaptation), and that PI$^2$ and CMA-ES thus share a common core. An obvious extension of our current work is to include other BBO algorithms in the empirical comparison. For instance, we would certainly expect CMA-ES to outperform its degenerate sibling PI$^{BB}$ on these tasks, and thus, by extension, also PI$^2$. In fact, it is likely that a large amount of BBO algorithms are able to outperform PI$^{BB}$ on the tasks considered. In this article, we have refrained from including these BBO methods in our comparison, to be able to specifically focus on the difference that arises when keeping all algorithmic features the same, except being RL or BBO methods.

# 5 Discussion

So why, on these five tasks, is PI$^2$ outperformed by the much simpler BBO algorithm PI$^{BB}$? It is rather counter-intuitive that an algorithm that uses *less* information is able to converge as fast or quicker than an algorithm that uses more information. This intuition is captured well in the following quote from Moriarty et al. (1999) *"In this sense, EA [BBO] methods pay less attention to individual decisions than TD [RL] methods do. While at first glance, this approach appears to make less efficient use of information, it may in fact provide a robust path toward learning good policies."*

In this discussion section, we first describe previous comparisons of RL and BBO algorithms, and then explain how our results extend the knowledge obtained in this previous work. In particular, we re-consider the trends in Figure 1 and Figure 7. We also discuss how the results we have obtained are influenced by the choice of policy representation and tasks used by Theodorou et al. (2010) and ourselves, which are tailored to the domain of learning skills on physical robots.

## 5.1 Previous Work on Empirically Comparing RL and BBO

The earliest empirical comparison of RL and BBO that we are aware of is the work of Moriarty et al. (1999). They compare "Evolutionary Algorithms for Reinforcement Learning"

[6] In this article, we show that removing covariance matrix adaptation from CMA-ES reduces it to PI$^{BB}$: "PI$^{BB}$ = CMA-ES minus CMA". In an orthogonal line of research (Stulp and Sigaud, 2012), we demonstrated the advantages of adding CMA-ES-style covariance matrix updating to PI$^2$, which yields the PI$^2$-CMAES algorithm: "PI$^2$-CMAES = PI$^2$ plus CMA". For a discussion of the advantages of adding covariance matrix updating in RL and BBO, we refer to (Stulp and Sigaud, 2012).

(EARL) with Q-learning on a simple MDP grid world, and conclude that these two methods "while complementary approaches, are by no means mutually exclusive." and that BBO approaches are advantageous "in situations where the sensors are inadequate to observe the true state of the world." (Moriarty et al., 1999).

Heidrich-Meisner and Igel (2008b) compare the performance of CMA-ES and NAC on a single pole balancing task. They conclude that *"Our preliminary comparisons indicate that the* CMA-ES *is more robust w.r.t. to the choice of hyperparameters and initial policies. In terms of learning speed, the natural policy gradient ascent performs on par for fine-tuning and may be preferable in this scenario."* This work was later extended to a double pole balancing task (Heidrich-Meisner and Igel, 2008a), where similar conclusions were drawn. A more extensive evaluation on single- and double pole balancing tasks is performed by (Gomez et al., 2008). They also conclude that *"in real world control problems, neuroevolution [. . . ] can solve these problems much more reliably and efficiently than non-evolutionary reinforcement learning approaches"*.

In an extensive comparison, Rückstiess et al. (2010b) compare PGPE, REINFORCE, NES, eNAC, NES and CMA-ES one pole-balancing, biped standing, object grasping, and ball catching. Their focus is particularly on comparing action-perturbing and parameter-perturbing algorithms. Their main conclusion is that parameter-perturbation outperforms action-perturbation: *"We believe that parameter-based exploration should play a more important role not only for PG methods but for continuous RL in general, and continuous value-based RL in particular"* (Rückstiess et al., 2010b).

An issue with such comparisons is that *"each of these efforts typically only compares a few algorithms on a single problem, leading to contradictory results regarding the merits of different RL methods."* (Togelius et al., 2009). It is this issue that we referred to in the introduction: if CMA-ES outperforms eNAC on a particular task, is it because of their different perturbation methods, their different parameter update methods, or because one is BBO and the other is RL? One of the main goals of this article is to provide an algorithmic framework that allows us to specifically investigate the latter question, whilst keeping the other algorithmic features the same.

Kalyanakrishnan and Stone (2011) aim at *"characterizing reinforcement learning methods through parameterized learning problems"*. They compare Sarsa, ExpSarsa, Q-learning, CEM and CMA-ES on problems consisting of simple square grids with a finite number of states. One interesting conclusion is that they are able to partially corroborate several of the conjectures by Togelius et al. (2009). The main difference to previous work is that *"our parameterized learning problem enables us to evaluate the effects of individual parameters while keeping others fixed."* (Kalyanakrishnan and Stone, 2011). Our work is orthogonal to this, in that it provides a pair of algorithms in which experimenters may switch between BBO and RL, whilst keeping other algorithmic features fixed. We thus focus on 'parameterizable algorithms', rather than parameterizable learning problems.

## 5.2 Reconsidering the Observed Trends

We now reconsider and evaluate the trends in Figure 7, given the empirical results presented in this article.

### 5.2.1 From Gradient-based Methods to Reward-weighted averaging

We believe the trend from gradient-based methods to reward-weighted averaging to have been an important step in enabling policy improvement methods to become robust towards noisy, discontinuous cost functions. From a theoretical perspective, we find it striking that reward-weighted averaging may be derived from fundamental principles in a wide variety of domains: reinforcement learning (Kober and Peters, 2011), stochastic optimal con-

trol (Theodorou et al., 2010), rare-event theory (Rubinstein and Kroese, 2004), and a basic set of optimality principles in BBO (Arnold et al., 2011). In practice, two algorithms that use this principle in BBO (CMA-ES) and RL ($\text{PI}^2$) turn out to be state-of-the-art in terms of empirical performance.

Previous work has focussed on comparing BBO methods that use reward-weighted averaging with gradient-based RL methods (Heidrich-Meisner and Igel, 2008a,b; Rückstiess et al., 2010b). This, however, is a rather unfair comparison, as RL methods based on reward-weighted averaging — such as PoWER and $\text{PI}^2$— have been shown to substantially outperform gradient-based RL methods (Kober and Peters, 2011; Theodorou et al., 2010). The reason such comparisons have not yet been made is that PoWER and $\text{PI}^2$ have only been introduced recently. To the best of our knowledge, this work is the first in comparing RL and BBO algorithms that are both based on reward-weighted averaging. Our conclusion is that BBO ($\text{PI}^{\text{BB}}$) is still able to outperform reward-weighted averaging RL ($\text{PI}^2$) on the tasks considered, but the margin is much smaller than when comparing BBO with gradient-based RL (e.g. eNAC). We expect this margin to increase again when using more sophisticated BBO algorithms, such as CMA-ES, than $\text{PI}^{\text{BB}}$. This is part of our current work.

#### 5.2.2 From Action Perturbation to Parameter Perturbation

Going from action perturbation to parameter perturbation seems to have been a fruitful trend, as confirmed by Rückstiess et al. (2010b). Mapping action perturbations to parameter updates requires a mapping from action space to parameter space, and requires knowledge of the derivative of the policy. In contrast, parameter perturbing methods perform exploration in the same space as in which the parameter update takes place. Thus, the controlled variable that leads to variations in the cost is also directly the variable that will be updated. Empirically, algorithms based on parameter perturbation substantially outperform those based on action perturbation (Rückstiess et al., 2010b; Heidrich-Meisner and Igel, 2008a; Theodorou et al., 2010).

#### 5.2.3 From Rewards at Each Time Step to Aggregrated Costs

Although $\text{PI}^{\text{BB}}$, which uses only a scalar aggregrated cost, outperforms $\text{PI}^2$, which uses the costs at each time step, on the tasks presented by Theodorou et al. (2010) and used in this article, we do not hold this to be a general result. We believe the cause must lie in the chosen tasks themselves, or the particular policy representation we have chosen. These tasks and representations in their turn are biased by our particular interest in applying policy improvement to acquire robotic skills. In fact, we have (informally) compared $\text{PI}^{\text{BB}}$ and $\text{PI}^2$ on several other robotic tasks not reported here, and have not found one instance where $\text{PI}^2$ outperforms $\text{PI}^{\text{BB}}$. Thus, understanding why BBO outperforms RL on these types of tasks may be related to understanding if and how the properties of typical tasks and policy representations used in robotic skill learning make them particularly amenable to BBO. Kalyanakrishnan and Stone (2011): "[T]he relationships between problem instances and the performance properties of algorithms are unclear, it becomes a worthwhile pursuit to uncover them". The results presented in this article are a first step in the pursuit to uncover the relationship between typical robotic tasks and the performance properties of BBO/RL. This topic, of particular interest to roboticists, is at the center of our current investigations.

### 5.3 Relation to the Conjectures by Togelius et al. (2009)

Our experiments, as do those of (Kalyanakrishnan and Stone, 2011), corroborate several of the conjectures by Togelius et al. (2009).

**Continuous state and actions** *"[BBO] method [...] generally outperform [RL] methods on problems with continuous state spaces".* We have studied only continuous state and action spaces, and BBO has equal or better performance than RL, which is a corroboration of this conjecture.

**Intermediate rewards** *Since [RL] methods, unlike [BBO] method, can use all experiential information obtained during interaction with the environment, [RL] methods outperform [BBO] algorithms in applications where it is helpful to exploit intermediate rewards ... [quote continued below]"* (Togelius et al., 2009). For the tasks in (Theodorou et al., 2010), exploiting intermediate rewards hardly plays a role. Therefore, we have added an extra task in which three via-points must be passed through. However, for this task, BBO ($PI^{BB}$) also converges much faster than RL ($PI^{2-}$), and to the same final cost (cf. Figure 13). We cannot corroborate this conjecture.

**Short, episodic tasks** *[Quote continued from above] ... especially if episodes are long."* Our research interest is learning skills for robots, with a particular focus on manipulation (Stulp et al., 2012; Marin et al., 2011). Typical robotic skills for manipulation — reaching for an object, transporting it to another location — typically do not take longer than about a second. This also holds for the simulated tasks described in Appendix A. For such short tasks, it is less likely that intermediate rewards play an important role.

# 6 Conclusion

Based on four algorithmic properties, we have provided a classification of policy improvement algorithms. By defining these properties concisely and clearly, one of the results of this article is to distinguish between RL and BBO based on these properties alone, rather than specific algorithms. For instance, we argued that action perturbing methods cannot use a BBO update rule. Furthermore, algorithms that require information about states/actions in the trajectories arising from policy roll-outs, such as actor-critic methods, cannot be BBO algorithms. Also, although finite-differencing methods (FD) are often to be considered an RL approach to policy improvement, it must be acknowledged that the algorithm may *also* be interpreted as being a BBO algorithm, given the definition of a BBO problem.

A second result is that, within this classification, we observe three trends in the chronology and derivation paths of algorithms: from gradient-based methods to reward-weighted averaging, from action to parameter perturbation, and towards algorithms that use only reward information from policy roll-outs.

We have continued this trend by applying two simplifications to the $PI^2$ algorithm: 1) keep exploratory parameter perturbations constant during a roll-out; 2) eliminate temporal averaging by considering only the entire cost of the trajectory, rather than the cost-to-go at each time step. This leads to a novel, much simpler algorithm, called $PI^{BB}$. We show that $PI^{BB}$ is a BBO algorithm, and a specific degenerate case of CMA-ES.

In previous work, it was shown that $PI^2$ is able to outperform PEGASUS, REINFORCE, and eNAC and PoWER (Theodorou et al., 2010). Using exactly the same tasks, we observe rather surprisingly that the much simpler BBO algorithm $PI^{BB}$ has equal or better performance than $PI^2$ still. Previous work on comparing RL and BBO shows that BBO often wins by a wide margin; the caveat being that, in those experiments, the BBO methods use reward-weighted averaging whereas the RL methods use gradient estimation. An important conclusion of our results is that the margin, though still existent, is much smaller when *both* RL and BBO are based on the powerful concept of reward-weighted averaging.

Although BBO thus trumps RL on several tasks, we do not believe this to be a general result, and further investigations are needed, especially into the bias that typical tasks and policy representations used in robotics — the types used in this article — introduce into RL problems.

Rather than making the case for BBO or RL, one of the main contributions of this article is to provide an algorithmic framework in which such cases may be made. Because $\text{PI}^{\text{BB}}$ and $\text{PI}^2$ use identical perturbation and parameter update methods, and differ only in being BBO and RL approaches respectively, this allows for a more objective comparison of BBO and RL than for instance comparing algorithms that differ in many respects. Therefore, we believe this algorithmic pair is an excellent basis for comparing BBO and RL approaches to policy improvement, and further investigating the five conjectures in (Togelius et al., 2009).

# Acknowledgements

We would like to thank Mrinal Kalakrishnan, Jonas Buchli, Nikolaus Hansen and Balázs Kégl for fruitful discussions and suggestions for improvement. A special thanks to Matthieu Geist for proofreading an earlier version of the article. We thank Stefan Schaal for providing the source code for running the experiments and tasks in (Theodorou et al., 2010). This work is supported by the French ANR program MACSi (ANR 2010 BLAN 0216 01), http://macsi.isir.upmc.fr

# A Evaluation Tasks

In this section, we describe the tasks used for the empirical evaluations in Section 3.1.3 and 3.2.2. These tasks are taken from the article by Theodorou et al. (2010). The implementations are based on the same source code as in Theodorou et al. (2010), and all tasks and algorithms parameters are the same unless stated otherwise. This allows for a direct comparison of the results in this article and those acquired by Theodorou et al. (2010). Due to the similarity, this appendix is very similar to Section 5 of (Theodorou et al., 2010), and added for completeness only.

## A.1 DMP and $\text{PI}^2$ Parameterization

In all the tasks below, the DMPs have 10 basis functions per dimension, and a duration of 0.5s. During learning, $K = 15$ roll-outs are performed for one update. Although 10 roll-outs has usually proven to be sufficient, Theodorou et al. (2010) choose 15 roll-outs to allow comparison with eNAC, which requires at least 1 roll-out more than the number of basis functions to perform its matrix inversion without numerical instabilities. The initial exploration magnitude is $\lambda = 0.05$ for all tasks except Task 1, where it is $\lambda = 0.01$. The exploration decay, which was tuned separately for each task, is 0.98, 0.98, 0.99, 0.99, 0.999 for Task 1. . . 5 respectively.

## A.2 Task 1

This task considers a 1-dimensional DMP of duration 0.5s, which starts at $x_0 = 0$ and ends at the goal $g = 1$. In this task as in all others, the initial movement is acquired by training the DMP with a minimum-jerk movement. The aim of Task 1 is to reach the goal $g$ with high accuracy, whilst minimizing acceleration, which is expressed with the following immediate ($r_t$) and terminal ($\phi_{t_N}$) costs:

$$r_t = 0.5 f_t^2 + 5000 \boldsymbol{\theta}^{\intercal} \boldsymbol{\theta}, \quad \phi_{t_N} = 10000(\dot{x}_{t_N}^2 + 10(g - x_{t_N})^2) \tag{40}$$

where $f_t$ refers to the linear spring-damper system in the DMP, cf. (29). Figure 15 visualizes the movement before and after learning.

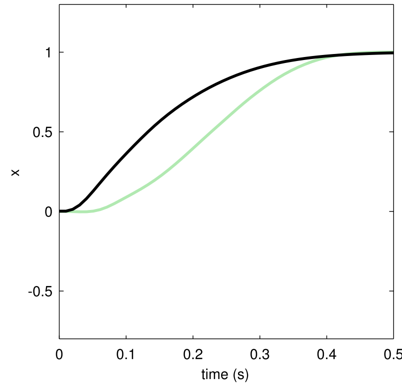


Figure 15: Task 1: Reaching the goal accurately whilst minimizing accelerations before (light green) and after (black) learning.

## A.3 Task 2 & 3

In Task 2, the aim is for the output of the 1-dimensional DMP (same parameters as in Task 1) to pass through the viapoint $0.25$ at time $t = 300ms$. Which is expressed with the costs:

$$r_{300ms} = 10^8(0.25 - x_{t_{300ms}})^2, \quad \phi_{t_N} = 0 \tag{41}$$

The costs are thus 0 at each time step except at $t_{300ms}$. This cost function was chosen by Theodorou et al. (2010) to allow for the design of a compatible function for PoWER.

Task 3 is equivalent except that it uses 3 viapoints [0.5 -0.5 1.0] at times [100ms 200ms 300ms] respectively. Figure 16 visualizes the movement before and after learning for Task 2 and Task 3.

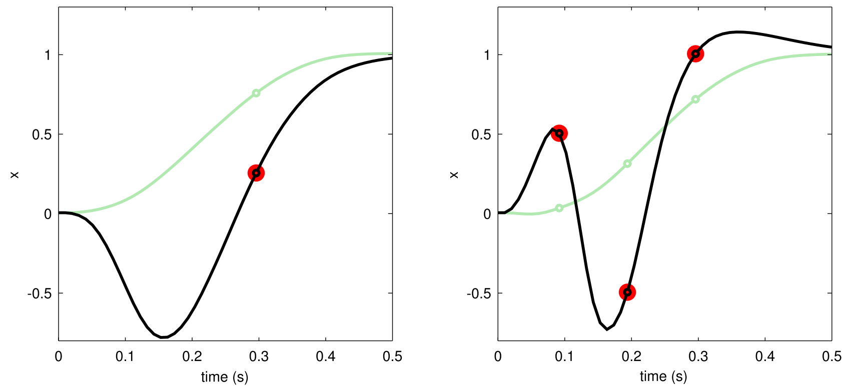


Figure 16: Task 2 (left) and 3 (right): Minimizing the distance to 1 or 3 viapoints before (light green) and after (black) learning.

Note that Task 3 was not evaluated by Theodorou et al. (2010). We have included it as we expected that it is a task where it may be *"helpful to exploit intermediate rewards"* (Togelius et al., 2009), and where RL approaches are conjectured to outperform BBO (Togelius et al., 2009). As Figure 13 reveals, this is not the case for this particular task, and $\mathrm{PI}^{\mathrm{BB}}$ also outperforms $\mathrm{PI}^2$ for this task.

## A.4 Task 4 & 5

Theodorou et al. (2010) used this task to evaluate the scalability of PI$^2$ to high-dimensional action spaces and learning problems with high redundancy. Here, an 'arm' with $D$ rotational joints and $D$ links of length $\frac{1}{D}$ is kinematically simulated in 2D Cartesian space. Figure 17 visualizes the movement by showing the configuration of the arm at each time step. The goal is again to pass through a viapoint (0.5,0.5) , this time in end-effector space, whilst minimizing accelerations. The $D$ joint trajectories are initialized with a minimum-jerk trajectory, and then optimized with respect to the following cost function:

$$r_t = \frac{\sum_{i=1}^{D}(D+1-i)(0.1f_{i,t}^2 + 0.5\boldsymbol{\theta}_i^{\intercal}\boldsymbol{\theta}_i)}{\sum_{j=1}^{D}(D+1-j)} \tag{42}$$

$$r_{300ms} = 10^8((0.5 - x_{t_{300ms}})^2 + (0.5 - y_{t_{300ms}})^2) \tag{43}$$

$$\phi_{t_N} = 0 \tag{44}$$

The weighting term $(D + 1 - i)$ places more weight on proximal joints than distal ones, which is motivated by the fact that proximal joints have lower mass and therefore less inertia, and are therefore more efficient to move (Theodorou et al., 2010). Figure 17 depicts the movements before and after learning for arms with $D = 2$ and $D = 10$ links respectively.

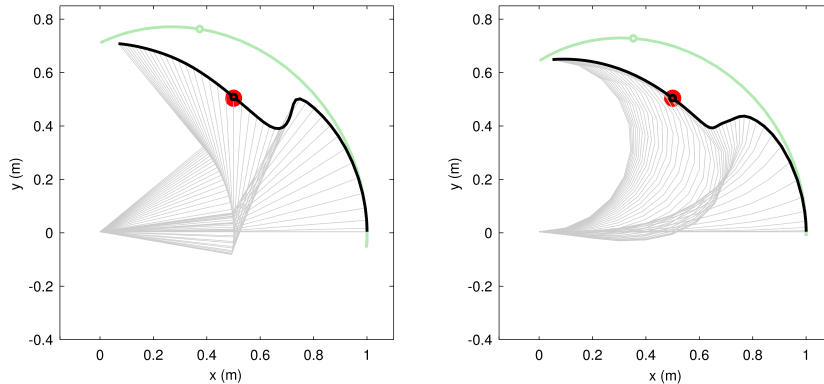


Figure 17: Task 4 (left, 2-DOF) and 5 (right, 10-DOF): minimizing the distance to a viapoint in end-effector space whilst minimizing joint accelerations.

# References

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

J. Buchli, F. Stulp, E. Theodorou, and S. Schaal. Learning variable impedance control. *International Journal of Robotics Research*, 30(7):820–833, 2011. URL .

L. Busoniu, D. Ernst, B. D. Schutter, and R. Babuska. Cross-entropy optimization of control policies with adaptive basis functions. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 41(1):196–209, 2011.

J. Fix and M. Geist. Monte-carlo swarm policy search. In *Symposium on Swarm Intelligence and Differential Evolution*, Lecture Notes in Artificial Intelligence (LNAI), page 9 pages. Springer Verlag - Heidelberg Berlin, Zakopane (Poland), 2012. URL .

F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9:937–965, 2008.

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

V. Heidrich-Meisner and C. Igel. Evolution strategies for direct policy search. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X*, pages 428–437, Berlin, Heidelberg, 2008a. Springer-Verlag. ISBN 978-3-540-87699-1. . URL .

V. Heidrich-Meisner and C. Igel. Similarities and differences between policy gradient methods and evolution strategies. In *ESANN 2008, 16th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 23-25, 2008, Proceedings*, pages 149–154, 2008b. .

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.

S. Kalyanakrishnan and P. Stone. Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning*, 84(1-2):205–247, 2011.

H. Kappen. Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(11):P11011, 2005. URL .

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

D. Marin and O. Sigaud. Towards fast and adaptive optimal control policies for robots: A direct policy search approach. In *Proceedings Robotica*, pages 21–26, Guimaraes, Portugal, 2012.

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 (GECCO'11)*, pages 1235–1242. ACM Press, 2011.

D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence (JAIR)*, 11:241–276, 1999.

A. Y. Ng and M. I. Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000. ISBN 1-55860-709-9. URL .

J. Peters and S. Schaal. Applying the episodic natural actor-critic architecture to motor primitive learning. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 1–6, 2007.

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, May 2008a. ISSN 0893-6080. . URL .

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

W. B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. Wiley-Blackwell, 2007.

M. Riedmiller, J. Peters, and S. Schaal. Evaluation of Policy Gradient Methods and Variants on the Cart-Pole Benchmark. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 254–261. IEEE, Apr. 2007. ISBN 1-4244-0706-0. . URL .

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

T. Rückstiess, M. Felder, and J. Schmidhuber. State-dependent exploration for policy gradient methods. In *19th European Conference on Machine Learning (ECML)*, 2010a.

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, 2010b. ISSN 2080-9778.

J. Santamaría, R. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163–217, 1997.

R. Stengel. *Optimal Control and Estimation*. Dover Publications, New York, 1994.

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

F. Stulp, E. Theodorou, and S. Schaal. Reinforcement learning with sequences of motion primitives for robust manipulation. *IEEE Transactions on Robotics*, 2012. Accepted for publication.

R. Sutton and A. Barto. *Reinforcement Learning: an Introduction*. MIT Press, 1998.

M. Tamosiumaite, B. Nemec, A. Ude, and F. Wörgötter. Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robots and Autonomous Systems*, 59(11):910–922, 2011.

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.

J. Togelius, T. Schaul, D. Wierstra, C. Igel, F. Gomez, and J. Schmidhuber. Ontogenetic and phylogenetic reinforcement learning. *Zeitschrift Künstliche Intelligenz - Special Issue on Reinforcement Learning*, pages 30–33, 2009.

S. Whiteson and P. Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, May 2006.

D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural evolution strategies. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, 2008.

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