Path Integral Policy Improvement with Differential Dynamic Programming

Tom Lefebvre[1,2,*] and Guillaume Crevecoeur[1,2]

***Abstract*— Path Integral Policy Improvement with Covariance Matrix Adaptation (PI^2-CMA) is a step-based model-free reinforcement learning approach that combines statistical estimation techniques with fundamental results from Stochastic Optimal Control. Basically, a policy distribution is improved iteratively using reward weighted averaging of the corresponding rollouts. It was assumed that PI^2-CMA somehow exploited gradient information that was contained by the reward weighted statistics. To our knowledge we are the first to expose the principle of this gradient extraction rigorously. Our findings reveal that PI^2-CMA essentially obtains gradient information similar to the forward and backward passes in the Differential Dynamic Programming (DDP) method. It is then straightforward to extend the analogy with DDP by introducing a feedback term in the policy update. This suggests a novel algorithm which we coin Path Integral Policy Improvement with Differential Dynamic Programming (PI^2-DDP). The resulting algorithm is similar to the previously proposed Sampled Differential Dynamic Programming (SaDDP) but we derive the method independently as a generalization of the framework of PI^2-CMA. Our derivations suggest to implement some small variations to SaDDP so to increase performance. We validated our claims on a robot trajectory learning task.**

I. Introduction

Policy Improvement with Path Integrals (PI^2) is a recent step-based model-free continuous state-action reinforcement learning method that was pioneered by [1]. PI^2 boils down to iteratively improving the mean of a policy distribution using reward weighted averaging of the corresponding system rollouts. The algorithm emerges naturally from the framework of linearly solvable optimal control [2], [3]. It makes a connection between value function approximating by solving a statistical inference problem based on empirical system rollouts, and direct policy learning [3], [4].

The striking similarity of PI^2 with Cross Entropy optimization methods, such as the Covariance Matrix Adaption Evolutionary Strategy (CMA-ES) [5], was first noticed by [6]. Their findings suggested a new algorithm, named PI^2-CMA, that additionally adapts the covariance of the policy distribution, claiming that the covariance matrix evolves to a matrix that is proportional to the inverse of the problem's Hessian [5]. The resulting algorithm can be categorized as an action perturbed reward weighted averaging approach [7].

To our knowledge we are the first to document that there is a rigorous connection between above reward weighted strategy and conventional gradient based methods. Essentially, the reward weighted statistics of the rollouts turn out to be directly related to the derivates of the value function similarly to the forward and backward pass in Differential Dynamic Programming (DDP) [8], [9]. Based on our findings it is straightforward to deepen the analogy with DDP and extend the PI^2-CMA algorithm with a feedback mechanism. Given analogy with preceding frameworks, we coin it PI^2-DDP.

It turns out that this algorithm is similar to Sampled Differential Dynamic Programming (SaDDP). An algorithm that was derived independently in [10]. The authors followed a more intuitive approach and did not expose the gradient extraction mechanism rigorously. Our findings suggest some minor adaptations that improve convergence nonetheless.

The article proceeds as follows. First we set-up the framework of linearly solvable Optimal Control (OC), which is a special class of OC problems. Within this framework, the PI^2 presents itself almost naturally. In following sections we derive and validate the proposed PI^2-DDP.

II. Background

A. Linearly solvable optimal control

Consider a nonlinear control affine continuous state-action Markov Decision Process with Gaussian transition probability, π. Here $\mathbf{x} \in \mathrm{R}^n$ denotes the state and $\mathbf{u} \in \mathrm{R}^m$ the control action while $\mathbf{a}$ and B define the afine map. Throughout we use $'$ to indicate discrete time propagation.

$$\mathbf{x}' \sim \pi = \mathcal{N}(\mathbf{a}(\mathbf{x}) + \mathrm{B}(\mathbf{x})\mathbf{u}, \Sigma) \tag{1}$$

We are interested in finite horizon optimal control problems which may be expressed using the Bellman equation (2) for given cost rate, l, and terminal cost, m, at final time N. Here v represents the value function, j the time instant and $\mathrm{E}_p\{\cdot\}$ the expected value operator w.r.t. the density p.

$$v_j(\mathbf{x}) = \min_{\mathbf{u}} l(\mathbf{x}, \mathbf{u}) + \mathrm{E}_\pi\{v_{j+1}(\mathbf{x}')\},\ v_N(\mathbf{x}) = m(\mathbf{x}) \tag{2}$$

Within described setting, the concept of linearly solvable optimal control is best understood when we reformulate the problem above slightly. Rather than searching for an optimal policy $\mathbf{u}^*$, we take interest in the optimal transition probability π^*. Note that both problems are equivalent since $\pi^* \triangleq \mathcal{N}(\mathbf{a}+\mathrm{B}\mathbf{u}^*, \Sigma)$. It has been shown [2] that if actuation is penalized using the Kullback-Leibler (KL) divergence (D_{KL}) between the actuated and free transition probabilities, respectively π and $p = \mathcal{N}(\mathbf{a}, \Sigma)$, there exists a value function transformation that linearises (2). Note that for Gaussian probabilities the divergence reduces to a quadratic control rate, $\frac{1}{2}\mathbf{u}^\top \mathrm{R}\mathbf{u}$, where $\mathrm{R} \equiv \mathrm{B}^\top\Sigma^{-1}\mathrm{B}$, the so called *linear solvability criterion*. This condition is understood as governing the signal-to-noise ratio, i.e. the admissible controls are similar in magnitude to any disturbances.

$$l(\mathbf{x}, \pi) = q(\mathbf{x}) + D_{\mathrm{KL}}(\pi \parallel p) = q(\mathbf{x}) + \tfrac{1}{2}\mathbf{u}^\top \mathrm{B}^\top\Sigma^{-1}\mathrm{B}\mathbf{u} \tag{3}$$

The value function will then satisfy

$$w(\mathbf{x}, j) = \exp(-q(\mathbf{x}))\mathrm{E}_p\{w(\mathbf{x}', j+1)\} \tag{4}$$

with $w \triangleq \exp(-v)$.

[1]Department of Electrical Energy, Metals, Mechanical Constructions & Systems, Ghent University, B-9052 Ghent, Belgium.

[2]EEDT Decision and Control, Flanders Make.

[*]Corresponding author: tom.lefebvre@ugent.be

The optimal π^* satisfies [11]

$$\pi^*(\mathbf{x}'|\mathbf{x}) = \frac{\mathcal{N}(\mathbf{x}'|\mathbf{a},\Sigma)w'(\mathbf{x}')}{\int \mathcal{N}(\mathbf{y}|\mathbf{a},\Sigma)w'(\mathbf{y})\mathrm{d}\mathbf{y}} \tag{5}$$

Above relation is mathematically elegant but not practical. For further development, we are interested in recovering an expression for the optimal policy $\mathbf{u}^*$ instead of π^*. We may invoke expression (5) and recall that by definition $\pi^* = \mathcal{N}(\mathbf{a}+\mathrm{B}\mathbf{u}^*,\Sigma)$. The idea is now simply to compare the expected value of both expressions. Since that $\mathbf{a}$ is independent of $\mathbf{y}$ and π is a probability density function, we can place $\mathbf{a}$ under the integral. It follows that

$$\mathbf{a}+\mathrm{B}\mathbf{u}^* = \int \mathbf{y}\pi^*(\mathbf{y}|\mathbf{x})\mathrm{d}\mathbf{y} \Rightarrow \mathrm{B}\mathbf{u}^* = \frac{\int(\mathbf{y}-\mathbf{a})w'(\mathbf{y})\mathcal{N}(\mathbf{a},\Sigma)\mathrm{d}\mathbf{y}}{\int w'(\mathbf{y})\mathcal{N}(\mathbf{a},\Sigma)\mathrm{d}\mathbf{y}}$$

Remark that the inverse of B is ill defined so that we have to use a pseudo inverse. We find that only $\mathrm{B}^\dagger = \mathrm{R}^{-1}\mathrm{B}^\top\Sigma^{-1}$ yields a satisfying[1] result. The resulting policy $\mathbf{u}^*$ satisfies

$$\mathbf{u}^* = \mathrm{R}^{-1}\mathrm{B}^\top\Sigma^{-1}\frac{\mathrm{E}_p\{(\mathbf{x}'-\mathbf{a})w'\}}{\mathrm{E}_p\{w'\}} \tag{6}$$

This peculiar result suggests that we can estimate policy $\mathbf{u}^*$ from system rollouts obtained with the free dynamics p.

B. Policy Improvement with Path Integrals

Kappen et al. were the first to realise that (6) can be practised to compute an optimal policy by means of statistical estimation techniques. Their approach can be considered as the first Policy Improvement with Path Integrals (PI^2) but was quite sample intensive and so scalability became an issue. In [1], Theodoreau et al. suggested to iterate the procedure so that with each iteration a better estimate is retrieved. As a result sampling could be coarser. Their procedure boils down to the following. In the first generation, K rollouts are generated with free dynamics $\mathbf{a}$ to obtain a first policy estimate $\mathbf{u}^{(1)}$. In future generations $g+1$, the free dynamics are updated as $\mathbf{a}^{(g+1)} \leftarrow \mathbf{a}+\mathrm{B}\mathbf{u}^{(g+1)}$ where $\mathbf{u}^{(g+1)}$ is updated as $\mathbf{u}^{(g+1)} \leftarrow \mathbf{u}^{(g)}+\mathbf{k}^{(g)}$ and the process is repeated. Here $\mathbf{k}^{(g)}$ is like a feedforward update and is calculated according to (6). Modifying the free dynamics also requires to update the cost, $q^{(g+1)} \leftarrow q+\frac{1}{2}\mathbf{u}^{(g+1)\top}\mathrm{R}\mathbf{u}^{(g+1)}$.

We emphasize that within the PI^2 framework, the stochastic policy perturbation is deliberate since the aim is to learn from random trials where the noise affects the action space. The linear solvability criterion then reduces to $\mathrm{B}^\top(\mathrm{B}\Sigma_\mathbf{u}\mathrm{B}^\top)^{-1}\mathrm{B}^\top = \mathrm{R}$. In order to enforce linear solvability we have $\Sigma_\mathbf{u} \triangleq \mathrm{R}^{-1}$. One can observe that for generation $g+1$, described process is equivalent with sampling $\mathbf{u}$ from $\mathcal{N}(\mathbf{u}_j^{(g)}+\mathbf{k}_j^{(g)},\Sigma_\mathbf{u})$ and updating the policy as $\mathbf{u}^{(g+1)} = \frac{1}{K}\sum_k \mathbf{u}_j^k \approx \mathbf{u}_j^{(g)}+\mathbf{k}_j^{(g)}$. This seems to overcomplicate matters but will turn out to yield insight later on.

As mentioned, the feedforward $\mathbf{k}^{(g)}$ is calculated using (6)

$$\mathbf{k}_j^{(g)} = \mathrm{R}^{-1}\mathrm{B}_j^\top\Sigma_j^{-1}\frac{\mathrm{E}_p^{(g)}\left\{\left(\mathbf{x}'-\mathbf{a}_j^{(g)}\right)w_{j+1}\right\}}{\mathrm{E}_p^{(g)}\{w_{j+1}\}} \tag{7}$$

[1]To see this, substitute (3) in (2), express the first order optimality criteria and solve for $\mathbf{u}$ yielding $\mathbf{u} = -\mathrm{R}^{-1}\mathrm{B}^\top\Sigma^{-1}\frac{\int(\mathbf{y}-\mathbf{a})\mathcal{N}(\mathbf{a}+\mathrm{B}\mathbf{u},\Sigma)v'(\mathbf{y})\mathrm{d}\mathbf{y}}{1-\int\mathcal{N}(\mathbf{a}+\mathrm{B}\mathbf{u},\Sigma)v'(\mathbf{y})\mathrm{d}\mathbf{y}}$, (we use the linear solvability criteria) suggesting that $\mathrm{B}^\dagger = \mathrm{R}^{-1}\mathrm{B}^\top\Sigma^{-1}$.

Repeated application of the recursion rule for w (4) and estimating the expected value operators with rollouts, yields following approximate expressions. Here s is defined as $\exp(-r(\boldsymbol{\tau}_{j+1:N}^k))$ and r is the cost-to-go function, operating on rollouts k encoded by $\boldsymbol{\tau}_{j:N}^k = \{\mathbf{x}_{j:N}^k, \mathbf{u}_{j:N}^k\}$, see (10).

$$\mathrm{E}_p\{w_{j+1}\} \approx \tfrac{1}{K}\textstyle\sum_k s\left(\boldsymbol{\tau}_{j+1:N}^k\right) \tag{8}$$

and (with $\Delta = \mathrm{R}^{-1}\mathrm{B}^\top\Sigma^{-1}(\mathbf{y}-\mathbf{a})$)

$$\mathrm{E}_p\{\Delta_j \cdot w_{j+1}\} \approx \tfrac{1}{K}\textstyle\sum_k \mathrm{D}_j^k\mathbf{d}_j^k s\left(\boldsymbol{\tau}_{j+1:N}^k\right) \tag{9}$$

where $\mathrm{D} = \mathrm{R}^{-1}\mathrm{B}^\top\Sigma^{-1}$ and $\mathbf{d}_j = \mathbf{x}_{j+1}-\mathbf{a}_j$.

The cost-to-go function is defined as

$$r\left(\boldsymbol{\tau}_{j:N}^k\right) = m(\mathbf{x}_N^k) + \textstyle\sum_{l=j}^{N-1} q(\mathbf{x}_l^k) + \sum_{l=j}^{N-1} \frac{1}{2}\mathbf{u}_l^{k\top}\mathrm{R}\mathbf{u}_l^k \tag{10}$$

Finally remark that $\mathbf{d}_j^k = \mathrm{B}_j^k \cdot \delta\mathbf{u}_j^k$, where $\delta\mathbf{u}_j^k = \mathbf{u}_j^k - \mathbf{u}_j^{(g)}$ since the drift between observed and expected dynamics is caused by the noise and that $\mathrm{DB} = \mathrm{R}^{-1}\mathrm{B}^\top\Sigma^{-1}\mathrm{B} = \mathrm{I}$ considering that R and Σ are related according to $\mathrm{B}^\top\Sigma^{-1}\mathrm{B} = \mathrm{R}$ as these results only hold true in the linearly solvable context.

These results can be combined into the following expression for the time dependent updates $\mathbf{k}_j^{(g)}$

$$\mathbf{k}_j^{(g)} = \textstyle\sum_k \frac{s_{j+1}^k}{\sum_k s_{j+1}^k}\delta\mathbf{u}_j^k = \sum_k p_{j+1}^k\delta\mathbf{u}_j^k \tag{11}$$

where s_j^k is shorthand notation for $s(\boldsymbol{\tau}_{j:N}^k)$.

In the current form this algorithm may be interpreted as a reward weighted averaging of random policies. To our knowledge this is also the first discrete time derivation of the PI^2 algorithm as the derivation in [1] was within a continuous time setting. It is interesting to note that the continuous time version [1] initiates the weights (11) at j rather than at $j+1$. We may argue that the policy at time instant j will only affect future time steps and that hence the weights should only contain future information.

Algorithm 1 discrete time PI^2-CMA

initialise $\left\{\mathbf{u}_j^{(0)}, \mathbf{k}_j^{(0)}\right\}_{j=0}^{N-1} \equiv 0$ and $\left\{\Sigma_{\mathbf{u},j}^{(0)}\right\}_{j=0}^{N-1} \equiv \mathrm{R}^{-1}$
for $g = 1, 2, 3, ...$ **do**
  for $k = 1, 2, 3, ..., K$ **do**
    set $\mathbf{x}_0^k = \mathbf{x}_0$
    for $j = 0, 1, 2, ..., N-1$ **do**
      sample $\mathbf{u}_j^k$ from $\mathcal{N}\left(\mathbf{u}_j^{(g)}+\mathbf{k}_j^{(g)}, \Sigma_{\mathbf{u},j}^{(g)}\right)$
      collect running cost $q_j^k + \frac{1}{2}\mathbf{u}_j^{k\top}\mathrm{R}\mathbf{u}_j^k$
    end for
    collect terminal cost m_N^k
  end for
  calculate cost-to-go values $\left\{\tilde{r}_j^k\right\}_{k=1,j=1}^{K,N}$
  calculate probability weights $\left\{p_j^k\right\}_{k=1,j=1}^{K,N}$
  for $j = 0, 1, 2, ..., N-1$ **do**
    $\mathbf{k}_j^{(g+1)} \leftarrow \sum_k p_{j+1}^k\delta\mathbf{u}_j^k$
    $\Sigma_j^{(g+1)} \leftarrow \sum_k p_{j+1}^k\delta\mathbf{u}_j^k\delta\mathbf{u}_j^{k\top}$
    $\mathbf{u}_j^{(g+1)} \leftarrow \frac{1}{K}\sum_k \mathbf{u}_j^k$
  end for
end for

C. Improvements

Since the algorithm was first introduced, two improvements have been made. Firstly, an optimal baseline can be defined w.r.t. the cost-to-go values r_j^k. So to differentiate *good* from *bad* random updates, one modifies r_j^k to

$$\tilde{r}_j^k = \beta \frac{r_j^k - \min r_j^k}{\max r_j^k - \min r_j^k} \tag{12}$$

where β is a sensitivity parameter.

The latter modification was already suggested by [1]. More recently, Stulp et al. [6] proposed to update the policy perturbation covariance, $\Sigma_{\mathbf{u}}$, according to

$$\Sigma_{\mathbf{u},j}^{(g+1)} \leftarrow \sum\nolimits_k p_{j+1}^k \delta\mathbf{u}_j^k \delta\mathbf{u}_j^{k\top} \tag{13}$$

based on structural similarities with the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) (see [5] for a good introduction). Supposedly $\Sigma_{\mathbf{u},j}^{(g+1)}$ would then converge to a matrix proportional to the inverse Hessian of the value function, the learning rate improved correspondingly.

These modifications result into PI^2-CMA. This method was originally intended for parametrised policies. We refrain from those here. Pseudocode is provided in algorithm 1.

III. Derivation of PI^2-DDP from first principles

In this section we detail the derivation of PI^2-DDP. First we provide a brief resume of the standard DDP method. Next we elaborate on the gradient extraction mechanism that, as we found, is inherent to PI^2-CMA. Based on these preliminary results, we will then derive PI^2-DDP.

A. Differential Dynamic Programming

First consider the following definition of the Q-function

$$Q_j(\mathbf{x}, \mathbf{u}) = l(\mathbf{x}, \mathbf{u}) + \mathrm{E}_\pi\{v_{j+1}(\mathbf{x}')\} \tag{14}$$

Now assume that we dispose of a nominal policy $\mathbf{u}^{(g)}$ and corresponding trajectory $\mathbf{x}^{(g)}$. In proximity of these nominal trajectories, we may approximate the Q-function using a second order Taylor model. Note that we abbreviate the couple $\{\mathbf{x}, \mathbf{u}\}$ to the state-action $\boldsymbol{\tau}$, for notational brevity.

$$\Delta Q(\Delta\boldsymbol{\tau}) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \Delta\boldsymbol{\tau} \end{bmatrix}^\top \begin{bmatrix} 0 & \mathbf{g}_{\boldsymbol{\tau}}^\top \\ \mathbf{g}_{\boldsymbol{\tau}} & \mathrm{H}_{\boldsymbol{\tau}} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\boldsymbol{\tau} \end{bmatrix} \tag{15}$$

Here $\Delta\boldsymbol{\tau}$ represents a local deviation between a given and the nominal trajectory, i.e. $\boldsymbol{\tau}$ and $\boldsymbol{\tau}^{(g)} = \{\mathbf{x}^{(g)}, \mathbf{u}^{(g)}\}$. Here $\mathbf{g}_{\boldsymbol{\tau}}$ and $\mathrm{H}_{\boldsymbol{\tau}}$ are defined as the gradient and Hessian of the Q-function (partial derivatives are indicated with subscripts)

$$\mathbf{g}_{\boldsymbol{\tau}} = \begin{bmatrix} \mathbf{g}_{\mathbf{x}} \\ \mathbf{g}_{\mathbf{u}} \end{bmatrix} \qquad \mathrm{H}_{\boldsymbol{\tau}} = \begin{bmatrix} \mathrm{H}_{\mathbf{xx}} & \mathrm{H}_{\mathbf{xu}} \\ \mathrm{H}_{\mathbf{ux}} & \mathrm{H}_{\mathbf{uu}} \end{bmatrix} \tag{16}$$

The key principle in DDP is to solve for $\Delta\mathbf{u}$ as a function of an unknown $\Delta\mathbf{x}$, i.e. $\Delta\mathbf{u} = -\mathrm{H}_{\mathbf{uu}}^{-1}\mathbf{g}_{\mathbf{u}} - \mathrm{H}_{\mathbf{uu}}^{-1}\mathrm{H}_{\mathbf{ux}}\Delta\mathbf{x}$. Note that this is equal to a simultaneous feedforward and -backward update, say $\Delta\mathbf{u} = \mathbf{k} + \mathrm{K}\Delta\mathbf{x}$. Based on this idea an iterative scheme can be designed. In a forward step, one updates the policy as

$$\mathbf{u}_j^{(g+1)} = \mathbf{u}_j^{(g)} + \mathbf{k}_j^{(g)} + \mathrm{K}_j^{(g)} \left(\mathbf{x}_j^{(g+1)} - \mathbf{x}_j^{(g)} \right) \tag{17}$$

whilst in a backward recursion step, estimates for the Hessian and gradient are obtained in proximity of the updated nominal state-action trajectory, $\boldsymbol{\tau}^{(g+1)} = \{\mathbf{x}^{(g+1)}, \mathbf{u}^{(g+1)}\}$.

Alternatively, if we abstain from the forward propagation step, we set the feedback gain $\mathrm{K} = 0$, which yields the following update (which basically corresponds with a second order single shooting method).

$$\mathbf{u}_j^{(g+1)} = \mathbf{u}_j^{(g)} + \mathbf{k}_j^{(g)} \tag{18}$$

The resemblance with PI^2-CMA is remarkable. Moreover, assessing the construction of $\mathbf{k}_j^{(g)}$ in (11) and considering that $\delta\mathbf{u}_j^k$ are distributed according the inverse Hessian, its plausible to assume that their exists a one-on-one relation between (11) and DDP feedforward $\mathbf{k}_j^{(g)} = -\mathrm{H}_{\mathbf{uu}}^{-1}\mathbf{g}_{\mathbf{u}}$.

B. Mechanism of gradient extraction in PI^2-CMA

Following the derivation of the PI^2-CMA in section II-B, we found that the policy update was the outcome of an optimization procedure at time instant j. Consequently the weights only contained information about future time steps.

In order to establish a rigorous connection with gradient based algorithms, we require the weights to contain information about the current time instant as well. Henceforth we shall consider the modified PI^2 update with time shifted weights (as is standard in continuous time PI^2) and focus on the state-action vector rather than on the policy alone

$$\delta\boldsymbol{\tau}_j \leftarrow \sum\nolimits_k \frac{s\left(\boldsymbol{\tau}_{j:N}^k\right)}{\sum_k s\left(\boldsymbol{\tau}_{j:N}^k\right)} \delta\boldsymbol{\tau}_j^k \tag{19}$$

Let us first consider the normalizers. According to the linearly solvable optimal control framework, it holds that

$$\tfrac{1}{K} \sum\nolimits_k s\left(\boldsymbol{\tau}_{j:N}^k\right) \approx \exp\left(-Q_j(\tilde{\boldsymbol{\tau}}_j)\right) \tag{20}$$

Here the nominal trajectory $\tilde{\boldsymbol{\tau}}$ can be estimated from the system rollouts as $\mathrm{E}\{\boldsymbol{\tau}\} \approx \frac{1}{K}\sum_k \boldsymbol{\tau}^k$ (that is equal to $\boldsymbol{\tau}^{(g+1)}$ *after* executing generation $g+1$). Remark that it would be particularly convenient if we could somehow rewrite the numerators as a function of Q as well. We can do this using a trick, rewriting the term $\frac{1}{K}s^k$ as a difference of sums

$$\begin{aligned} \tfrac{1}{K}s^k &= \tfrac{1}{K}\textstyle\sum_{k=1}^K s^k - \tfrac{1}{K}\sum_{l=1,l\neq k}^K s^l \\ &\approx \exp\left(-Q\left(\tilde{\boldsymbol{\tau}}\right)\right) - \tfrac{K-1}{K}\exp\left(-Q\left(\tilde{\boldsymbol{\tau}}^k\right)\right) \end{aligned} \tag{21}$$

where $\tilde{\boldsymbol{\tau}}^k$ is now estimated as $\frac{1}{K-1}\sum_{l\neq k}\boldsymbol{\tau}^l$.

Combination of these results reveals that the weights, p^k, contain an approximated Q-function difference

$$\begin{aligned} p^k &\approx 1 - \tfrac{K-1}{K}\exp\left(Q\left(\tilde{\boldsymbol{\tau}}\right) - Q\left(\tilde{\boldsymbol{\tau}}^k\right)\right) \\ &\approx 1 - \tfrac{K-1}{K}\exp\left(-\Delta Q\left(\Delta\tilde{\boldsymbol{\tau}}^k\right)\right) \end{aligned} \tag{22}$$

Here $\Delta\tilde{\boldsymbol{\tau}}^k$ is the difference between the nominal trajectory estimates, i.e. $\tilde{\boldsymbol{\tau}}^k - \tilde{\boldsymbol{\tau}}$. Note that from the definitions of $\tilde{\boldsymbol{\tau}}$ and $\tilde{\boldsymbol{\tau}}^k$ it follows that $K\tilde{\boldsymbol{\tau}} - (K-1)\boldsymbol{\tau} = \tilde{\boldsymbol{\tau}}^k$. Rearranging for $\tilde{\boldsymbol{\tau}}^k$ yields $\tilde{\boldsymbol{\tau}}^k = -\frac{1}{K-1}\boldsymbol{\tau}^k + \frac{K}{K-1}\tilde{\boldsymbol{\tau}}$ so that we have that $\Delta\tilde{\boldsymbol{\tau}}^k = -\frac{1}{K-1}\delta\boldsymbol{\tau}^k$. Clearly we can estimate the function difference using a second order approximation

$$\begin{aligned}\Delta Q\left(\Delta\tilde{\boldsymbol{\tau}}^k\right) &\approx \Delta Q\left(-\tfrac{1}{\gamma}\delta\boldsymbol{\tau}^k\right)\\ &= \frac{1}{2}\begin{bmatrix}1\\ \delta\boldsymbol{\tau}^k\end{bmatrix}^\top\begin{bmatrix}0 & -\frac{1}{\gamma}\mathbf{g}_{\boldsymbol{\tau}}^\top\\ -\frac{1}{\gamma}\mathbf{g}_{\boldsymbol{\tau}} & \frac{1}{\gamma^2}\mathrm{H}_{\boldsymbol{\tau}}\end{bmatrix}\begin{bmatrix}1\\ \delta\boldsymbol{\tau}^k\end{bmatrix}\end{aligned} \tag{23}$$

where we introduce $\gamma = K - 1$ for notational brevity.

Now we invoke that by definition the sum of all weights, p^k, should be equal to 1 and substitute above relations into the identity $1 = \sum_k p^k$. Abbreviating $\delta\boldsymbol{\tau}^k$ to $\boldsymbol{\xi}^k$, and assuming that $\boldsymbol{\xi}$ is sampled from some Gaussian $\mathcal{N}(0, \Psi_{\boldsymbol{\tau}})$, then yields that, where $\mathrm{B}_{\boldsymbol{\tau}} = (\Psi_{\boldsymbol{\tau}}^{-1} + \frac{1}{\gamma^2}\mathrm{H}_{\boldsymbol{\tau}})^{-1} \approx \Psi_{\boldsymbol{\tau}}$

$$\begin{aligned}1 &\approx \sum_k \left(1 - \tfrac{\gamma}{1+\gamma}\exp\left(-\Delta Q\left(-\tfrac{1}{\gamma}\boldsymbol{\xi}^k\right)\right)\right)\\ &\approx (1+\gamma) - \gamma\int \exp\left(-\Delta Q\left(-\tfrac{1}{\gamma}\boldsymbol{\xi}\right)\right)\mathcal{N}(0,\Psi_{\boldsymbol{\tau}})\mathrm{d}\boldsymbol{\xi}\\ &= (1+\gamma) - \eta\cdot\gamma\int \mathcal{N}\left(\tfrac{1}{\gamma}\mathrm{B}_{\boldsymbol{\tau}}\mathbf{g}_{\boldsymbol{\tau}}, \mathrm{B}_{\boldsymbol{\tau}}\right)\mathrm{d}\boldsymbol{\xi}\end{aligned} \tag{24}$$

Remark that the integral is equal to 1 which on its turn implies that the normalizer η simply vanishes. These observations are key to reveal the principle of gradient extraction as we found is inherent to the reward weighted statistics.

Let us first consider the (extended) update $\delta\boldsymbol{\tau}$, it follows

$$\begin{aligned}\delta\boldsymbol{\tau} &= \textstyle\sum_k p^k\delta\boldsymbol{\tau}^k\\ &\approx (1+\gamma)\mathrm{E}_{\mathcal{N}(0,\Psi_{\boldsymbol{\tau}})}\{\boldsymbol{\xi}\} - \gamma\mathrm{E}_{\mathcal{N}(\frac{1}{\gamma}\mathrm{B}_{\boldsymbol{\tau}}\mathbf{g}_{\boldsymbol{\tau}},\mathrm{B}_{\boldsymbol{\tau}})}\{\boldsymbol{\xi}\}\\ &\approx -\mathrm{B}_{\boldsymbol{\tau}}\mathbf{g}_{\boldsymbol{\tau}} \approx -\Psi_{\boldsymbol{\tau}}\mathbf{g}_{\boldsymbol{\tau}}\end{aligned} \tag{25}$$

Similarly, for the (extended) covariance update, we find

$$\begin{aligned}\Sigma_{\boldsymbol{\tau}}^{(g+1)} &= \textstyle\sum_k p^k\delta\boldsymbol{\tau}^k\delta\boldsymbol{\tau}^{k\top}\\ &\approx \gamma\mathrm{E}_{\mathcal{N}(0,\Psi_{\boldsymbol{\tau}})}\left\{\boldsymbol{\xi}\boldsymbol{\xi}^\top\right\} - \tfrac{\gamma^2}{1+\gamma}\mathrm{E}_{\mathcal{N}(\frac{1}{\gamma}\mathrm{B}_{\boldsymbol{\tau}}\mathbf{g}_{\boldsymbol{\tau}},\mathrm{B}_{\boldsymbol{\tau}})}\left\{\boldsymbol{\xi}\boldsymbol{\xi}^\top\right\}\\ &\approx \gamma\Psi_{\boldsymbol{\tau}} - \tfrac{\gamma^2}{1+\gamma}\mathrm{B}_{\boldsymbol{\tau}} - \tfrac{1}{1+\gamma}\mathrm{B}_{\boldsymbol{\tau}}\mathbf{g}_{\boldsymbol{\tau}}\mathbf{g}_{\boldsymbol{\tau}}^\top\mathrm{B}_{\boldsymbol{\tau}}\end{aligned} \tag{26}$$

Here we used that $\frac{1}{1+\gamma}\sum\boldsymbol{\xi}^k\boldsymbol{\xi}^{k\top}$ yields biased estimates of the covariance, i.e. $\frac{\gamma}{1+\gamma}\Psi_{\boldsymbol{\tau}}$ instead of $\Psi_{\boldsymbol{\tau}}$.

C. PI²-DDP

In the previous section, we demonstrated how the PI²-CMA reward weighted rollout statistics are directly related to the gradient information of the local Q-function. This process is remarkably similar to the forward and backward passes of the DDP method. Dislike DDP, PI²-CMA does not engage a feedback strategy to update the policy. Based on exposed gradient extraction principle, we may introduce such a feedback term and update the policy according to the DDP direction in (17). In order to provide a smooth derivation we have to make two preliminary assumptions. The necessary conditions for those assumptions to hold true will emerge when we detail our calculations. The first assumption is that $\Psi_{\boldsymbol{\tau}} \approx \Sigma_{\boldsymbol{\tau}}^{(g)}$ and the second assumption is that $\Sigma_{\boldsymbol{\tau}}^{(g)} \approx \mathrm{H}_{\boldsymbol{\tau}}^{-1}$.

First let us attribute a similar block matrix structure to $\Sigma_{\boldsymbol{\tau}}^{(g)}$ in analogy with what we did earlier for $\mathrm{H}_{\boldsymbol{\tau}}$

$$\Sigma_{\boldsymbol{\tau}}^{(g)} = \begin{bmatrix}\Sigma_{\mathbf{xx}} & \Sigma_{\mathbf{xu}}\\ \Sigma_{\mathbf{ux}} & \Sigma_{\mathbf{uu}}\end{bmatrix} \tag{27}$$

Invoking the first assumption and (25) yields that

$$\mathbf{g}_{\boldsymbol{\tau}} \approx -{\Sigma_{\boldsymbol{\tau}}^{(g)}}^{-1}\delta\boldsymbol{\tau} \tag{28}$$

whilst according to the second assumption it holds that

$$\mathbf{g}_{\mathbf{u}} \approx -\mathrm{H}_{\mathbf{uu}}\delta\mathbf{u} - \mathrm{H}_{\mathbf{ux}}\delta\mathbf{x} \tag{29}$$

Substituting above expression in (17) delivers

$$\Delta\mathbf{u} \approx \delta\mathbf{u} + \mathrm{H}_{\mathbf{uu}}^{-1}\mathrm{H}_{\mathbf{ux}}\delta\mathbf{x} - \mathrm{H}_{\mathbf{uu}}^{-1}\mathrm{H}_{\mathbf{ux}}\Delta\mathbf{x} \tag{30}$$

where we may use that $\mathrm{H}_{\mathbf{uu}}^{-1}\mathrm{H}_{\mathbf{ux}} \approx -\Sigma_{\mathbf{ux}}\Sigma_{\mathbf{xx}}^{-1}$ due to the general block matrix inversion lemma [12], ultimately to arrive at the following feedback based update mechanism

$$\Delta\mathbf{u} = \underbrace{\delta\mathbf{u} - \Sigma_{\mathbf{ux}}\Sigma_{\mathbf{xx}}^{-1}\delta\mathbf{x}}_{\mathbf{k}} + \underbrace{\Sigma_{\mathbf{ux}}\Sigma_{\mathbf{xx}}^{-1}}_{\mathrm{K}}\Delta\mathbf{x} \tag{31}$$

1) Discussion of the first assumption: If we engage above update rule when we execute a learning batch, the following holds true for the rollout trajectories.

$$\boldsymbol{\tau}^k = \boldsymbol{\tau}^{(g+1)} + \boldsymbol{\xi}^k \rightarrow \begin{cases}\mathbf{x}^{(g+1)} + \boldsymbol{\zeta}^k\\ \mathbf{u}^{(g+1)} + \mathrm{K}^{(g)}\boldsymbol{\zeta}^k + \boldsymbol{\epsilon}^k\end{cases} \tag{32}$$

Recall that $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \Sigma_{\mathbf{u}})$ is a stochastic policy perturbation and $\boldsymbol{\zeta} \sim \mathcal{N}(0, \Psi_{\mathbf{xx}})$ is the accumulated deviation of the state due to perturbed actions at preceding time instants. Provisionally we assume that $\Psi_{\mathbf{xx}} \approx \Sigma_{\mathbf{xx}}$. It follows that

$$\begin{aligned}\Psi_{\mathbf{ux}} &= \mathrm{cov}\left\{\mathbf{u}\mathbf{x}^\top\right\} = \mathrm{Gcov}\left\{\boldsymbol{\zeta}\boldsymbol{\zeta}^\top\right\} = \Sigma_{\mathbf{ux}}\\ \Psi_{\mathbf{uu}} &= \mathrm{cov}\left\{\mathbf{u}\mathbf{u}^\top\right\} = \mathrm{cov}\left\{\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\right\} + \mathrm{Gcov}\left\{\boldsymbol{\zeta}\boldsymbol{\zeta}^\top\right\}\mathrm{G}^\top\\ &= \Sigma_{\mathbf{u}} + \Sigma_{\mathbf{ux}}\Sigma_{\mathbf{xx}}^{-1}\Sigma_{\mathbf{xu}}\end{aligned} \tag{33}$$

These observations reveal that the feedback mechanism itself is indispensable to obtain correlated state-action trajectories which on its turn is necessary since otherwise $\Psi_{\boldsymbol{\tau}}$ would be a block diagonal matrix and the update principle would collapse to PI²-CMA. To reassure that $\Psi_{\mathbf{uu}} \approx \Sigma_{\mathbf{uu}}$ holds true, note that $\Sigma_{\mathbf{u}} = \Sigma_{\mathbf{uu}} - \Sigma_{\mathbf{ux}}\Sigma_{\mathbf{xx}}^{-1}\Sigma_{\mathbf{xu}}$. The latter observation directly suggests following perturbation covariance matrix update scheme

$$\Sigma_{\mathbf{u},j}^{(g+1)} \leftarrow \Sigma_{\mathbf{uu}} - \Sigma_{\mathbf{ux}}\Sigma_{\mathbf{xx}}^{-1}\Sigma_{\mathbf{xu}} \tag{34}$$

It is left to prove that this procedure also generates state trajectories that are perturbed according to $\mathcal{N}(0, \Sigma_{\mathbf{xx}})$. Note that the initial state is not affected by the feedback mechanism, hence if left unattended its value is deterministic. This observation suggests that we should sample $\mathbf{x}_0^k$ from $\mathcal{N}(\mathbf{x}_0, \Sigma_{\mathbf{xx},0}^{(g)})$. We argue that this choice initiates an uncertainty propagation mechanism that generates state trajectories with the desired stochastic properties [13].

2) Discussion of the second assumption: To illustrate that $\Sigma_{\tau}^{(g)}$ converges to $\mathrm{H}_{\boldsymbol{\tau}}^{-1}$ we can simply substitute our first assumption in (26). In proximity of the optimum ($\mathbf{g}_{\boldsymbol{\tau}} \approx 0$), this generates a matrix sequence (26) that converges to $\lambda\mathrm{H}_{\boldsymbol{\tau}}^{-1}$ with λ converging to 0. To see this one may substitute $\lambda\mathrm{H}_{\boldsymbol{\tau}}^{-1}$ for $\Sigma_{\boldsymbol{\tau}}$. The generating matrix sequence then reduces to a scalar sequence for λ that satisfies

$$\lambda \leftarrow \gamma\lambda - \tfrac{\gamma^4}{(1+\gamma)(\lambda+\gamma^2)}\lambda \tag{35}$$

For $\lambda < \gamma^2/(\gamma^2-1)$ this sequence converges to 0. Close to 0, the convergence rate of λ is equal to $\gamma/(\gamma+1)$.

We may emphasize that the covariance matrix update scheme exhibits memory as is. Further damping of the covariance update scheme, as in state-of-the-art CMA Evolutionary Strategies [5], will be disadvantageous. This was observed empirically by the authors in [6]. A possible practice might be to damp the inverse Hessian estimates exponentially. This increases the convergence rate to $\frac{K-\alpha}{K}$.

$$\Sigma_{\boldsymbol{\tau}}^{(g+1)} = (1-\alpha)\Sigma_{\boldsymbol{\tau}}^{(g)} + \alpha \sum_k p^k \delta\boldsymbol{\tau}^k \delta\boldsymbol{\tau}^{k\top} \tag{36}$$

D. Remarks

Based on exposed gradient extraction principles we can deepen insight in the behaviour of PI^2 inspired methods.

Firstly, the use of the optimal base lining heuristic scales the derivative information contained by the reward weighted rollout statistics in section III-B by a factor $\beta/\Delta r$, which is equivalent to rescaling the optimization problem with the same factor. However, as we have demonstrated that PI^2-CMA and PI^2-DDP essentially behave as second order gradient algorithms, the policy updates are scale invariant and the choice of β is indeed relatively uncritical [6].

Secondly, it turned out that $\Sigma_{\boldsymbol{\tau}}^{(g)}$ converges to $\lambda \mathrm{H}_{\boldsymbol{\tau}}^{-1}$ rather than $\mathrm{H}_{\boldsymbol{\tau}}^{-1}$. This means that $\delta\boldsymbol{\tau} \approx -\lambda \mathrm{H}_{\boldsymbol{\tau}}^{-1}\mathbf{g}_{\boldsymbol{\tau}}$. As a result the update in (31) is scaled by a factor $\lambda < 1$. (Note that $\Delta\mathbf{x}$ depends on the perturbation covariance matrix, $\Sigma_{\mathbf{u}}^{(g)}$, which is also scaled by a factor λ (34).). Resultingly, λ seems to be a line-search parameter inherent to PI^2. In order to avoid that λ converges prematurely, we may superimpose a noise term to the update $\Sigma_{\mathbf{u},j}^{(g)}$, e.g. $\lambda_{\text{exp}}\mathrm{R}^{-1}$, as the perturbed policy is the only source of stochasticity. Note that λ_{exp} can be made adaptive in correspondence with the general convergence.

$$\Sigma_{\mathbf{u}}^{(g+1)} \leftarrow \Sigma_{\mathbf{uu}} - \Sigma_{\mathbf{ux}}\Sigma_{\mathbf{xx}}^{-1}\Sigma_{\mathbf{xu}} + \lambda_{\text{exp}}\mathrm{R}^{-1} \tag{37}$$

We may also choose to introduce a line search parameter scaling the feedforward as $\lambda_{\text{ls}}\mathbf{k}_j^{(g)}$. Also note that if we set $\mathrm{K} = 0$, (31) and (34) reduce to PI^2-CMA, since then the state-action rollouts become uncorrelated implying $\Sigma_{\mathbf{ux}} = 0$.

To prevent ill conditioning when we calculate the inverse matrices $\Sigma_{\mathbf{uu}}^{-1}$ and $\Sigma_{\mathbf{xx}}^{-1}$, required by the updates (11)-(34), we use a simple regularization heuristic, $\mathrm{A}^{\dagger} \approx (\mathrm{A} + \lambda_{\text{rg}}\mathrm{I})^{-1}$.

Pseudocode of the ensemble is provided in algorithm 2.

E. Previous derivation

The present algorithm was derived independently by the authors in [10], predating our derivation. The authors did however not detail the gradient extraction principles nor did they really present the algorithm as a natural extension of PI^2-CMA, and coined the method Sampled Differential Dynamic Programming (SaDDP). As a result they do not introduce a term $-\Sigma_{\mathbf{ux}}\Sigma_{\mathbf{xx}}^{-1}\delta\mathbf{x}_j^k$ to compensate for the gradient $\mathbf{g}_{\mathbf{x}}$ but proposed alternatively to bias $\Delta\mathbf{x}$ on $\mathbf{x}^{(g+1)}$ where $\mathbf{x}^{(g+1)}$ is updated according to $\mathbf{x}^{(g+1)} \leftarrow \mathbf{x}^{(g)} + \delta\mathbf{x}$. Remarkably, this is equivalent to (31) when $\lambda_{\text{fb}} = 1$. The authors also argued that optimal search directions were distributed according to $\mathcal{N}(0, \mathrm{H}_{\mathbf{uu}}^{-1})$ which resulted in the same update as in (34), we believe that it is simply necessary to sustain the covariance convergence scheme. Related to this last aspect, we also found that distributing the initial rollouts states, $\mathbf{x}_0^k$, according to $\mathcal{N}(\mathbf{x}_0^k, \Sigma_{\mathbf{xx},0}^{(g)})$ is necessary for the same reason. The latter suggestion was not included in [10].

IV. Numerical assessment

We have evaluated the PI^2 framework for a robot trajectory learning task. We focus the discussion on the learning rates of the different PI^2 inspired algorithms and detail some of the rationale behind our hyperparameter choices based on exposed gradient extraction principles.

A. Problem description

We set-up a small benchmark learning task. Our goal is to learn an optimal policy that manipulates a toy model of a PUMA 560 robot arm in between two end-effector positions, see Fig. 1. The policy corresponds with direct motor commands. Since it is generally hard to learn policies for unstable dynamics, we assumed perfect gravity compensation in our simulation environment. This is similar to assuming that we are generating reference trajectories to a low level controller. The robotic manipulator was modelled using general rigid-body dynamics, i.e. $\ddot{\mathbf{q}} = \mathrm{M}(\mathbf{q})^{-1}\mathbf{u} - \mathrm{M}(\mathbf{q})^{-1}\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$. The robot state is defined as $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}})$. Dynamics are discretised using Euler's method with a time step of $\Delta t = 20\text{ms}$.

We consider a finite horizon optimal control problem with time horizon $T = 0.5\text{s}$ and $N = 25$, terminal cost $m(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}_N^*\|_{\mathrm{M}}^2$, state cost rate $q(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}_N^*\|_{\mathrm{Q}}^2$ and control cost rate $\frac{1}{2}\|\mathbf{u}\|_{\mathrm{R}}^2$. Matrices M, Q and R are defined respectively as $10^9\mathrm{diag}(1,10,1,1,1,1)$, $\mathrm{diag}(5\cdot10^4, 10, 10, 10, 10, 10)$ and $\mathrm{diag}(10,10,10)$. Designing a well-shaped objective landscape was quite laborious and considered as a task that requires expertise. We will not elaborate on any of the design choices that we have made.

Algorithm 2 PI^2-DDP

initialise $\left\{\mathbf{u}_j^{(0)}\right\}_{j=0}^{N-1} \equiv 0$ and $\left\{\Sigma_{\mathbf{u},j}^{(0)}\right\}_{j=0}^{N-1} \equiv \mathrm{R}^{-1}$
for $g = 1, 2, 3, \ldots$ **do**
  for $k = 1, 2, 3, \ldots, K$ **do**
    sample $\mathbf{x}_0^k \sim \mathcal{N}\left(\mathbf{x}_0, \Sigma_{\mathbf{xx},0}^{(g)}\right)$
    for $j = 0, 1, 2, \ldots, N-1$ **do**
      sample $\mathbf{u}_j^k$ from
        $\mathcal{N}\left(\mathbf{u}_j^{(g)} + \lambda_{\text{ls}}\mathbf{k}_j^{(g)} + \mathrm{K}_j^{(g)}\left(\mathbf{x}_j^k - \mathbf{x}_j^{(g)}\right), \Sigma_{\mathbf{u},j}^{(g)}\right)$
      collect running cost $q_j^k + \frac{1}{2}\mathbf{u}_j^{k\top}\mathrm{R}\mathbf{u}_j^k$
    end for
    collect terminal cost m_N^k
  end for
  calculate cost-to-go values $\left\{\tilde{r}_j^k\right\}_{k=1,j=1}^{K,N}$
  calculate reward weights $\left\{p_j^k\right\}_{k=1,j=1}^{K,N}$
  for $j = 0, 1, 2, \ldots, N-1$ **do**
    $\Sigma_{\boldsymbol{\tau},j}^{(g+1)} \leftarrow (1-\alpha)\Sigma_{\boldsymbol{\tau},j}^{(g)} + \alpha\sum_k p_j^k \delta\boldsymbol{\tau}_j^k \delta\boldsymbol{\tau}_j^{k\top}$
    $\mathrm{K}_j^{(g+1)} \leftarrow \Sigma_{\mathbf{xu}}\Sigma_{\mathbf{xx}}^{\dagger}$
    $\mathbf{k}_j^{(g+1)} \leftarrow \sum_k p_j^k\left(\delta\mathbf{u}_j^k - \mathrm{K}_j^{(g+1)}\delta\mathbf{x}_j^k\right)$
    $\Sigma_{\mathbf{u},j}^{(g+1)} \leftarrow \Sigma_{\mathbf{uu}} - \Sigma_{\mathbf{ux}}\Sigma_{\mathbf{xx}}^{\dagger}\Sigma_{\mathbf{xu}} + \lambda_{\text{exp}}\mathrm{R}^{-1}$
    $\boldsymbol{\tau}_j^{(g+1)} \leftarrow \frac{1}{K}\sum \boldsymbol{\tau}_j^k$
  end for
end for

B. Setting of the hyperparameters

In order to choose β, one may consider that the optimal base-line heuristic in (12) alters the derivations in III-B in such a way that, $\delta\boldsymbol{\tau} \approx -\frac{\beta}{\Delta r}\Sigma_{\boldsymbol{\tau}}\mathbf{g}_{\boldsymbol{\tau}}$. In preliminary iterations where $\Sigma_{\boldsymbol{\tau}} \neq \mathrm{H}_{\boldsymbol{\tau}}^{-1}$, it acts like a line-search parameter. This means that β should be similar in magnitude to what can be expected of Δr. We set $\beta = 10^6$. Parameter α was set to $\frac{1}{2}$ as we found that larger values resulted into too noisy Hessian estimates and that smaller values corrupted the learning rate. The number of rollouts K should be large enough to assure enough information is contained within the rollouts to determine the gradient, i.e. $K > z = n + m$. Ideally K should exceed the information that is contained in both the gradient and the Hessian, i.e. $\frac{z^2}{2} + \frac{z}{2} + z$. We found this to be overly conservative given (26). We set $K = 15$.

C. Discussion

In Fig. 2, we provide mean learning rates for all PI^2 inspired algorithms for a total of 30 experiments. We make an additional distinction between **PI^2-DDP v1** and **PI^2-DDP v2**. The first version, like SaDDP, does not perturb $\mathbf{x}_0$ whilst the second version does according to $\mathcal{N}(0, \Sigma^{(g)}_{\mathbf{xx},0})$.

Note that we have made a distinction between the means of the 15 best and 15 worst attempts. That is because it turned out that all variations of the PI^2 framework are prone to a large variance in between distinct attempts. Unfortunately our simulation results do not reflect similar improvements as were documented in [10]. We may argue that the effect of feedback is less pronounced for this particular test case. Nonetheless one may observe that the **PI^2-DDP v2** outperforms all other variations and performs as **PI^2** in worst case.

We emphasize that λ_{exp} takes on an important role, given that it regulates the rate at which the covariance matrix vanishes. During our experiments we found that keeping λ_{exp} fixed may result in fast initial learning rates but degraded final convergence which is again in correspondence with III-D. Therefore we adopted following trust region inspired update heuristic. If the current exploration rate yields satisfying results, we increase the exploration magnitude slightly. In case that the policy is worse then the previous, we decrease its magnitude. Here $v^{(g)}$ represents the accumulated cost over a noise free rollout with feedback policy $\mathbf{u}^{(g)} + \mathrm{G}^{(g)}\Delta\mathbf{x}$.

$$\begin{aligned} &\textbf{if } v^{(g)} > v^{(g-1)}, && \lambda_{\text{exp}} \leftarrow \max\{10^{-3}, 0.9 \cdot \lambda_{\text{exp}}\} \\ &\textbf{else if } v^{(g)} < 0.9 \cdot v^{(g-1)}, && \lambda_{\text{exp}} \leftarrow \min\{1, 1.1 \cdot \lambda_{\text{exp}}\} \end{aligned} \tag{38}$$

In addition we can choose to reinitialize the next iteration with the old policy if $v^{(g)} > v^{(g-1)}$. This implies that we disregard information contained by an entire rollout batch.

Fig. 3 portrays mean learning curves for different implementations of the exploration heuristic in (38). The implementation referred to as **adaptive v1** indicates that exploration is adaptive but the policy is updated anyhow, the **adaptive v2** implementation additionally reinitialized the policy to the policy from the previous generation. Following observations can be made. The fixed implementation exhibits significant oscillatory behaviour in later generations. It is clear that the magnitude of the noise becomes relatively too large w.r.t. to the objective landscape and the improvement procedure becomes more or less random. Secondly we can observe that there is still a high performance variance between the three considered exploration heuristics. Although this is significantly less the case for **adaptive v2**. Studying the evolution of λ_{exp} in Fig. 4 reveals that **adaptive v2** explores more aggressively during the early generations and settles down faster in later generations. As a result the performance variance is reduced significantly. So it appears to be a rewarding strategy to disregard a policy update entirely, therewith rendering an entire rollout batch useless, if the corresponding cost-to-go is larger than that of the previous generation. These preliminary results suggest that adaptive schemes may significantly improve the learning rate and may be considered in future work for all hyperparameters.

For visual confirmation, Fig. 5 provides snapshots of the learned optimal trajectory.

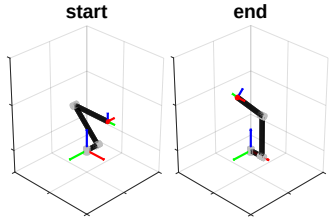

Fig. 1: Start and end positions for the learning task.

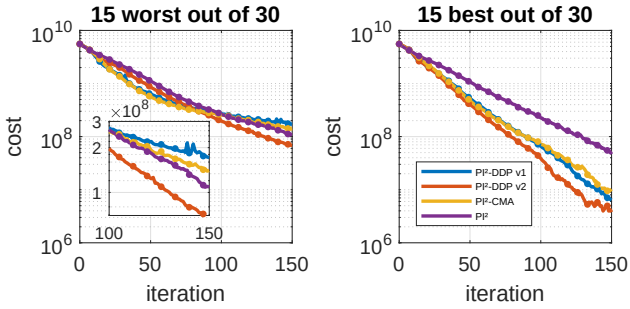

Fig. 2: Learning rates for all PI^2 algorithms, $\beta = 10^6$.

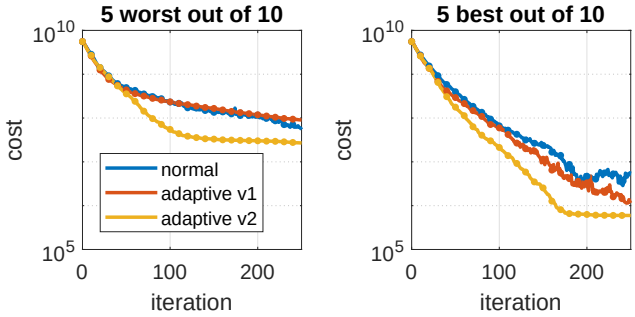

Fig. 3: Learning rates for different exploration strategies.

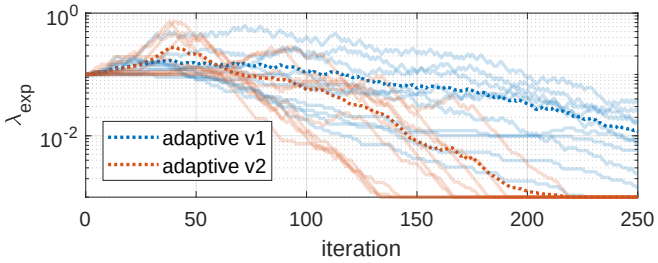

Fig. 4: Evolution of λ_{exp} for different exploration strategies.

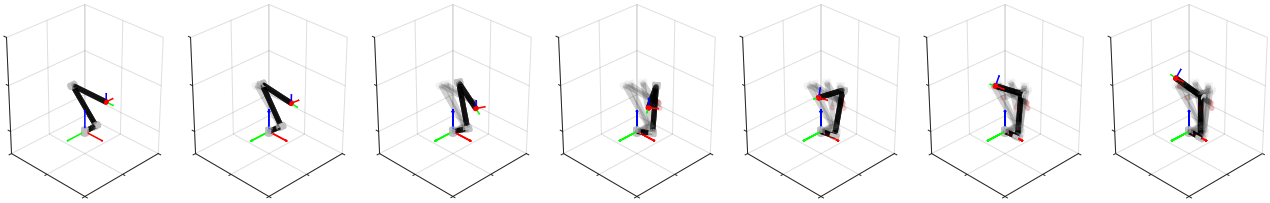

Fig. 5: Snapshots taken along the learned optimal trajectory.

V. Conclusion

We presented a step-based model-free reinforcement learning algorithm which we coined Path Integral Policy Improvement with Differential Dynamic Programming, connecting the Path Integral Policy Improvement with Covariance Matrix Adaptation (PI^2-CMA) method with the Differential Dynamic Programming (DDP) algorithm. Our main contribution is that we have demonstrated rigorously how first and second order derivative information of the value function is extracted by calculating the reward weighted statistics of system rollouts in accordance with PI^2-CMA.

This process is similar to the forward and backward passes in the state-of-the-art Differential Dynamic Programming (DDP) method. Based on these findings it is straightforward to develop an algorithm in analogy with DDP using the gradient extraction mechanism of PI^2-CMA. As a result we obtain a step-based model-free reinforcement learning algorithm with feedback based policy improvement. Our numerical experiments suggest that the introduction of a feedback term does improve the learning rate. We emphasize that the method is entirely model-free and hence has an attractive implementational threshold that will require limited storage capabilities of the embedded hardware.

The most important application is envisioned within the guided policy search framework [14] where recently the PI^2-CMA method was applied to learn a series of local controllers model-free [15]. Other applications might involve learning a local controller superimposed on a traditional model based controller.

Nevertheless our numerical experiments also indicate that the method is still prone to sources of stochasticity resulting in large performance variances, that need to be attended in future work. Amongst which are the complicated mechanism behind the covariance matrix adaptation (adopted from CMA) and the fact that our most performant exploration heuristic disregards the information contained by entire rollout batches that did not result into an improved policy. Future work should focus both in the theoretical assessment of the gradient extraction principle and on the development of improved covariance matrix adaptation schemes and more advanced adaptation schemes for the hyperparameters.

References

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

[2] E. Todorov. Linearly-solvable markov decision problems. In Advances in neural information processing systems, pages 1369–1376, 2007.

[3] H. Kappen, W. Wiegerinck, and B. van den Broek. A path integral approach to agent planning. Autonomous Agents and Multi-Agent Systems, 2007.

[4] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. In 2010 IEEE International Conference on Robotics and Automation, pages 2397–2403. IEEE, 2010.

[5] N. Hansen. The cma evolution strategy: A tutorial. arXiv preprint arXiv:1604.00772, 2016.

[6] F. Stulp and O. Sigaud. Path integral policy improvement with covariance matrix adaptation. arXiv preprint arXiv:1206.4621, 2012.

[7] Freek Stulp and Olivier Sigaud. Policy Improvement Methods: Between Black-Box Optimization and Episodic Reinforcement Learning. 34 pages, October 2012.

[8] W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In ICINCO (1), pages 222–229, 2004.

[9] D. Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. International Journal of Control, 3(1):85–95, 1966.

[10] J. Rajamäki, K. Naderi, V. Kyrki, and P. Hämäläinen. Sampled differential dynamic programming. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1402–1409, Oct 2016.

[11] E. Theodorou, D. Krishnamurthy, and E. Todorov. From information theoretic dualities to path integral and kullback-leibler control: Continuous and discrete time formulations. In The Sixteenth Yale Workshop on Adaptive and Learning Systems, 2013.

[12] K. Petersen, M. Pedersen, et al. The matrix cookbook. Technical University of Denmark, 7(15):510, 2008.

[13] E. Todorov. General duality between optimal control and estimation. In Decision and Control, 2008. CDC 2008. 47th IEEE Conference on, pages 4286–4292. IEEE, 2008.

[14] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In Advances in Neural Information Processing Systems, pages 1071–1079, 2014.

[15] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine. Path integral guided policy search. In 2017 IEEE international conference on robotics and automation (ICRA), pages 3381–3388. IEEE, 2017.