# Learning Policy Improvements with Path Integrals

**Evangelos Theodorou**
etheodor@usc.edu

**Jonas Buchli**
jonas@buchli.org

**Stefan Schaal**
sschaal@usc.edu

Computer Science, Neuroscience, & Biomedical Engineering
University of Southern California, 3710 S.McClintock Ave, Los Angeles, CA, 90089-2905
http://www-clmc.usc.edu

## Abstract

With the goal to generate more scalable algorithms with higher efficiency and fewer open parameters, reinforcement learning (RL) has recently moved towards combining classical techniques from optimal control and dynamic programming with modern learning techniques from statistical estimation theory. In this vein, this paper suggests the framework of stochastic optimal control with path integrals to derive a novel approach to RL with parametrized policies. While solidly grounded in value function estimation and optimal control based on the stochastic Hamilton-Jacobi-Bellman (HJB) equations, policy improvements can be transformed into an approximation problem of a path integral which has no open parameters other than the exploration noise. The resulting algorithm can be conceived of as model-based, semi-model-based, or even model free, depending on how the learning problem is structured. Our new algorithm demonstrates interesting similarities with previous RL research in the framework of probability matching and provides intuition why the slightly heuristically motivated probability matching approach can actually perform well. Empirical evaluations demonstrate significant performance improvements over gradient-based policy learning and scalability to high-dimensional control problems. We believe that **P**olicy **I**mprovement with **P**ath **I**ntegrals (**PI**$^2$) offers currently one of the most efficient, numerically robust, and easy to implement algorithms for RL based on trajectory roll-outs.

Appearing in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9. Copyright 2010 by the authors.

## 1 Introduction

While reinforcement learning (RL) is among the most general frameworks of learning control to create truly autonomous learning systems, its scalability to high-dimensional continuous state-action systems, e.g., humanoid robots, remains problematic. Classical value-function based methods with function approximation offer one possible approach, but function approximation under the non-stationary iterative learning process of the value-function remains difficult when one exceeds about 5-10 dimensions. Alternatively, direct policy learning from trajectory roll-outs has recently made significant progress (Peters, 2007), but can still become numerically brittle and full of open tuning parameters in complex learning problems. In new developments, RL researchers have started to combine the well-developed methods from statistical learning and empirical inference with classical RL approaches in order to minimize tuning parameters and numerical problems, such that ultimately more efficient algorithms can be developed that scale to significantly more complex learning system (Koeber and Peters, 2009, Peters and Schaal, 2008b, Toussaint and Storkey, 2006).

In the spirit of these latter ideas, this paper addresses a new method of probabilistic reinforcement learning derived from the framework of stochastic optimal control and path integrals, based on the original work of (Kappen, 2007, Broek et al., 2008). As will be detailed in the sections below, this approach makes an appealing theoretical connection between value function approximation using the stochastic HJB equations and direct policy learning by approximating a path integral, i.e., by solving a statistical inference problem from sample roll-outs. The resulting algorithm, called **P**olicy **I**mprovement with **P**ath **I**ntegrals (**PI**$^2$), takes on a surprisingly simple form, has no open tuning parameters besides the exploration noise, and performs numerically robustly in high dimensional learning problems. It also makes an interesting connection to previous work on RL based on probability matching (Dayan and Hinton, 1997, Peters and Schaal, 2008a, Koeber and Peters, 2009) and explains why probability matching algorithms can be successful.

## 2 Stochastic Optimal Control

In the analysis that follows we are making use of the control theoretic notation from trajectory-based optimal control, however, with an attempt to have as much overlap as possible with the standard RL notation (Sutton and Barto, 1998). We start our analysis by defining a finite horizon reward function for a trajectory $\boldsymbol{\tau}_i$ starting at time $t_i$ in state $\mathbf{x}_{t_i}$ and ending at time $t_N$

$$R(\boldsymbol{\tau}_i) = \phi_{t_N} + \int_{t_i}^{t_N} r_t \; dt \qquad (1)$$

with $\phi_{t_N} = \phi(x_{t_N})$ denoting a terminal reward at time $t_N$ and $r_t$ denoting the immediate reward at time $t$. In stochastic optimal control framework (Stengel, 1994), the goal is to find the controls $\mathbf{u}_t$ that minimize the value function:

$$V(\mathbf{x}_{t_i}) = V_t = \min_{\mathbf{u}_{t_i:t_N}} E_{\boldsymbol{\tau}_i}\left[R(\boldsymbol{\tau}_i)\right] \qquad (2)$$

where the expectation $E_{\boldsymbol{\tau}_i}[.]$ is taken over all trajectories starting at $\mathbf{x}_{t_i}$. We consider the rather general control system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}_t, t) + \mathbf{G}(\mathbf{x}_t)\left(\mathbf{u}_t + \epsilon_t\right) = \mathbf{f}_t + \mathbf{G}_t\left(\mathbf{u}_t + \epsilon_t\right) \quad (3)$$

with $\mathbf{x}_t \in \Re^{n\times 1}$ denoting the state of the system, $\mathbf{G}_t = \mathbf{G}(\mathbf{x}_t) \in \Re^{n\times p}$ the control matrix, $\mathbf{f}_t = \mathbf{f}(\mathbf{x}_t) \in \Re^{n\times 1}$ the passive dynamics, $\mathbf{u}_t \in \Re^{p\times 1}$ the control vector and $\epsilon_t \in \Re^{p\times 1}$ Gaussian noise with variance $\boldsymbol{\Sigma}_\epsilon$. As immediate reward we consider $r_t = r(\mathbf{x}_t, \mathbf{u}_t, t) = q_t + \mathbf{u}_t\mathbf{R}\mathbf{u}_t$ where $q_t = q(\mathbf{x}_t, t)$ is an arbitrary state-dependent reward function, and $\mathbf{R} > 0$ is weight matrix of the quadratic control cost with the inequality expressed in the positive definite sense. The HJB equation (Stengel, 1994, Fleming and Soner, 2006) associated with this stochastic optimal control problem is expressed as follows:

$$\partial_t V_t = q_t + (\partial_{\mathbf{x}} V_t)^T \mathbf{f}_t - \frac{1}{2}(\partial_{\mathbf{x}} V_t)^T \mathbf{G}_t \mathbf{R}^{-1}\mathbf{G}_t^T(\partial_{\mathbf{x}} V_t) \qquad (4)$$
$$+ \frac{1}{2}\mathbf{trace}\left((\partial_{\mathbf{xx}} V_t)\mathbf{G}_t\boldsymbol{\Sigma}_\epsilon\mathbf{G}_t^T\right)$$

The $\partial_{\mathbf{x}}$ and $\partial_{\mathbf{xx}}$ symbols refer to the Jacobian and Hessian, respectively, of the value function with respect to the state $\mathbf{x}$, while $\partial_t$ is the partial derivative with respect to time. The HJB is a second order nonlinear PDE. For notational compactness, we will mostly use subscripted symbols to denote time and state dependencies, as introduced in the equations above.

### 2.1 Exponential Transformation of HJB

In order to find a solution to the HJB equation above, we use a logarithmic transformation of the value function $V_t = -\lambda \log \Psi_t$ as well as the assumption $\lambda \mathbf{G}_t\mathbf{R}^{-1}\mathbf{G}_t^T = \mathbf{G}_t\boldsymbol{\Sigma}_\epsilon\mathbf{G}_t^T = \boldsymbol{\Sigma}(\mathbf{x}_t) = \boldsymbol{\Sigma}_t$. Thus, we obtain

$$-\partial_t\Psi_t = -\frac{1}{\lambda}q_t\Psi_t + \mathbf{f}_t^T(\partial_x\Psi_t) + \frac{1}{2}\mathbf{trace}\left((\partial_{\mathbf{xx}}\Psi_t)\mathbf{G}_t\boldsymbol{\Sigma}_\epsilon\mathbf{G}_t^T\right) \qquad (5)$$

with boundary condition: $\Psi_{t_N} = \exp\left(-\frac{1}{\lambda}\phi_{t_N}\right)$. The PDE in (5) corresponds to the so-called Kolmogorov backward PDE which is of second order and linear. Analytical solutions of (5) cannot be found in general for general nonlinear systems and cost functions. However, there is a connection between solutions of PDEs and their representation as stochastic differential equation (SDEs), which goes back to the Feynman-Kac formula (Oksendal, 2003),(Yong, 1997). The Feynman-Kac formula can be used to find distributions of random processes which solve certain SDEs as well as to propose numerical methods for solving certain PDEs.

According to the generalized version (Yong, 1997) of Feynman-Kac theorem: *Let $\mathbf{x}_t$ satisfies the following SDE $d\mathbf{x} = \mathbf{f}(\mathbf{x}_t, t)dt + \mathbf{G}(\mathbf{x}_t)d\omega$ with $t \in [t_i, t_N]$ where $\mathbf{x}(t_i) = \mathbf{x}_{t_i}$, $(t, \mathbf{x}) \in \Re \times \Re^{n\times 1}$ and $d\omega$ is a standard brownian motion.Then*

$$\Psi_{t_i} = E_{\boldsymbol{\tau}_i}\left(\Psi_{t_N} e^{-\int_{t_i}^{t_N}\frac{1}{\lambda}q_t dt}\right) \qquad (6)$$
$$= E_{\boldsymbol{\tau}_i}\left[\exp\left(-\frac{1}{\lambda}\phi_{t_N} - \frac{1}{\lambda}\int_{t_i}^{t_N} q_t \; dt\right)\right]$$

*If and only if $\Psi_{t_i}$ satisfies the Backward - Kolmogorov PDE:*

$$-\partial_t\Psi_t = -\frac{1}{\lambda}q_t\Psi_t + \mathbf{f}_t^T(\partial_x\Psi_t) + \frac{1}{2}\mathbf{trace}\left((\partial_{\mathbf{xx}}\Psi_t)\mathbf{G}_t\boldsymbol{\Sigma}_\epsilon\mathbf{G}_t^T\right) \qquad (7)$$

*with the boundary condition $\Psi_{t_N} = \exp\left(-\frac{1}{\lambda}\phi_{t_N}\right)$.*

This insight allows to represented the stochastic optimal control problem as an approximation problem of a path integral.

## 3 Generalized Path Integral Formulation

In many stochastic dynamical systems only some of the states are controlled, such that the state vector can be partitioned into $\mathbf{x} = [\mathbf{x}^{(m)^T} \quad \mathbf{x}^{(c)^T}]^T$ with uncontrollable part $\mathbf{x}^{(m)} \in \Re^{k\times 1}$ and controllable part $\mathbf{x}^{(c)} \in \Re^{l\times 1}$. Thus, the control variable $\mathbf{u} \in \Re^{p\times 1}$ has dimensionality smaller than the state, i.e., $p < n$. Subsequently, the passive dynamics term and the control transition matrix can be partitioned as $\mathbf{f}_t = [\mathbf{f}_t^{(m)^T} \quad \mathbf{f}_t^{(c)^T}]^T$ with $\mathbf{f}_m \in \Re^{k\times 1}$, $\mathbf{f}_c \in \Re^{l\times 1}$ and $\mathbf{G}_t = [\mathbf{0}_{k\times p} \quad \mathbf{G}_t^{(c)^T}]^T$ with $\mathbf{G}_t^c \in \Re^{l\times p}$. For such systems it can been shown that the solution (6) becomes:

$$\begin{aligned}\Psi_{t_j} &= \lim_{dt\to 0} \int \left(\Pi_{i=j}^{N-1} p\left(\mathbf{x}_{t_{i+1}}^{(c)} | \mathbf{x}_{t_i}\right)\right) \\ &\times \quad \exp -\frac{1}{\lambda}\left(\phi_{t_N} + \sum_{i=0}^{N-1} q_{t_i} dt\right) d\boldsymbol{\tau}_j\end{aligned} \tag{8}$$

Since $\boldsymbol{\tau}_j = \left(\mathbf{x}_{t_j}, \mathbf{x}_{t_{j+1}}, ....., \mathbf{x}_{t_N}\right)$ are sample paths starting at state $\mathbf{x}_{t_j}$, the integration above is taken with respect to $\boldsymbol{d\tau}_j = \left(\mathbf{dx}_{t_j}, ....., \mathbf{dx}_{t_N}\right)$. Assuming a Gaussian state transition probability:

$$p\left(\mathbf{x}_{t_{i+1}}^{(c)} | \mathbf{x}_{t_i}\right) = \left((2\pi)^l |\boldsymbol{\Sigma}_{t_i}|\right)^{-1/2} \exp\left(-\frac{1}{2\lambda}\gamma_{t_i} dt\right) \tag{9}$$

where $\gamma_{t_i} = \boldsymbol{\alpha}_{t_i}^T \mathbf{h}_{t_i}^{-1} \boldsymbol{\alpha}_{t_i}$, $\quad \mathbf{h}_{t_i} = \mathbf{G}_{t_i}^{(c)} \mathbf{R}^{-1} \mathbf{G}_{t_i}^{(c)T}$, and $\boldsymbol{\alpha}_{t_i} = \mathbf{x}_{t_{i+1}}^{(c)} - \mathbf{x}_{t_i}^{(c)} - \mathbf{f}_{t_i}^{(c)} dt$ . Substitution of the transition probability into (8) results in:

$$\Psi_{t_j} = \lim_{dt\to 0} \int e^{-\frac{1}{\lambda} Z(\boldsymbol{\tau}_j)} d\boldsymbol{\tau}_j \tag{10}$$

where $Z(\boldsymbol{\tau}_j) = S(\boldsymbol{\tau}_j) + \frac{\lambda}{2}\sum_{i=j}^{N-1} \log|\mathbf{h}_{t_i}| + \frac{\lambda N l}{2}\log(2\pi dt\lambda)$ and $S(\boldsymbol{\tau}_j) = -\phi_{t_N} - \frac{1}{2}\sum_{i=j}^{N-1}\gamma_{t_i} dt - \sum_{i=j}^{N} q_{t_i} dt$. By taking the limit as $dt \to 0$ we can calculate the logarithmic value function $\Psi_{t_j}$ which is the solution of (5) . The term $Z(\boldsymbol{\tau}_j)$ is the total cost of the sample path $\boldsymbol{\tau}_j$.

## 4 Optimal Controls

The optimal controls are given as $\mathbf{u}_{t_j} = -\mathbf{R}^{-1}\mathbf{G}_{t_j}^T(\partial_{x_{t_j}} V_{t_j})$. Due to the logarithmic transformation of the value function, the equation of the optimal controls can be written as $\mathbf{u}_{t_j} = \lambda \mathbf{R}^{-1}\mathbf{G}_{t_j}(\partial_{\mathbf{x}_{t_j}}\Psi_{t_j})/\Psi_{t_j}$. After substituting $\Psi_{t_j}$ with (10) and dropping the state independent terms of the cost we have:

$$\mathbf{u}_{t_j} = \lim_{dt\to 0}\left(\lambda \mathbf{R}^{-1}\mathbf{G}_{t_j}^T \frac{\partial_{\mathbf{x}_{t_j}}\left(\int e^{-\frac{1}{\lambda}\tilde{S}(\boldsymbol{\tau}_j)}\boldsymbol{d\tau}_j\right)}{\int e^{-\frac{1}{\lambda}\tilde{S}(\boldsymbol{\tau}_j)}\boldsymbol{d\tau}_j}\right) \tag{11}$$

with $\tilde{S}(\boldsymbol{\tau}_j) = S(\boldsymbol{\tau}_j) + \frac{\lambda}{2}\sum_{i=j}^{N-1}\log|\mathbf{h}_{t_j}|$. Further analysis of the equation above leads to a simplified version of the equation for optimal controls formulated as $\mathbf{u}_{t_j} = \int P(\boldsymbol{\tau}_j)\,\mathbf{u}(\boldsymbol{\tau}_j)\,\boldsymbol{d\tau}_j$ with the probability $P(\boldsymbol{\tau}_j)$ and local controls $\mathbf{u}(\boldsymbol{\tau}_j)$ defined as

$$P(\boldsymbol{\tau}_j) = \frac{e^{\frac{1}{\lambda}\tilde{S}(\boldsymbol{\tau}_j)}}{\int e^{\frac{1}{\lambda}\tilde{S}(\boldsymbol{\tau}_j)}\boldsymbol{d\tau}_j} \tag{12}$$

$$\mathbf{u}(\boldsymbol{\tau}_j) = -\mathbf{R}^{-1}\mathbf{G}_{t_j}^{(c)T}\lim_{dt\to 0}\left(\partial_{\mathbf{x}_{t_j}^{(c)}}\tilde{S}(\boldsymbol{\tau}_j)\right) \tag{13}$$

The path cost $\tilde{S}(\boldsymbol{\tau}_j)$ is a generalized version of the path cost in (Broek et al., 2008, Kappen, 2007), which only considered systems with state independent control transition. [1] To find the local controls $\mathbf{u}(\boldsymbol{\tau}_j)$ we have to calculate the $\lim_{dt\to 0}\partial_{\mathbf{x}_{t_j}^{(c)}}\tilde{S}(\boldsymbol{\tau}_j)$. Due to space limitations, we do not provide the detailed derivations. The final result is expressed as follows:

$$\lim_{dt\to 0}\left(\partial_{\mathbf{x}_{t_j}^{(c)}}\tilde{S}(\boldsymbol{\tau}_j)\right) = -\mathbf{h}_{t_j}^{-1}\left(\mathbf{G}_{t_j}^{(c)}\boldsymbol{\epsilon}_{t_j} - \mathbf{b}_{t_j}\right) \tag{14}$$

where the new term $\mathbf{b}_{t_j}$ is defined as $\mathbf{b}_{t_j} = \lambda \mathbf{h}_{t_j}\boldsymbol{\Phi}_{t_j}$ and $\boldsymbol{\Phi}_{t_j} \in \Re^{l\times 1}$ a vector with the *jth* element defined as:

$$\left(\boldsymbol{\Phi}_{t_j}\right)_j = trace\left(\mathbf{h}_{t_j}^{-1}\cdot\partial_{\mathbf{x}_{t_j}^{(cj)}}\mathbf{h}_{t_j}\right) \tag{15}$$

The local control can now be expressed as:

$$\mathbf{u}(\boldsymbol{\tau}_j) = \mathbf{R}^{-1}\mathbf{G}_{t_j}^{(c)T}\mathbf{h}_{t_j}^{-1}\left(\mathbf{G}_{t_j}^{(c)}\boldsymbol{\epsilon}_{t_j} - \mathbf{b}_{t_j}\right) \tag{16}$$

By substituting $\mathbf{h}_{t_j} = \mathbf{G}_{t_j}^{(c)}\mathbf{R}^{-1}\mathbf{G}_{t_j}^{(c)T}$ in the equation above we get our main result for the local controls of the sampled path for the generalized path integral formulation:

$$\mathbf{u}(\boldsymbol{\tau}_j) = \mathbf{R}^{-1}\mathbf{G}_{t_j}^{(c)T}\left(\mathbf{G}_{t_j}^{(c)}\mathbf{R}^{-1}\mathbf{G}_{t_j}^{(c)T}\right)^{-1}\left(\mathbf{G}_{t_j}^{(c)}\boldsymbol{\epsilon}_{t_j} - \mathbf{b}_{t_j}\right) \tag{17}$$

This is the generalized expression for the local controls and it includes special cases based on the properties of the control transition matrix involved in the stochastic dynamics. More precisely for systems with one dimensional controllable state space:

$$\mathbf{u}(\boldsymbol{\tau}_i) = \frac{\mathbf{R}^{-1}\mathbf{g}_{t_i}^{(c)}\mathbf{g}_{t_i}^{(c)T}}{\mathbf{g}_{t_i}^{(c)T}\mathbf{R}^{-1}\mathbf{g}_{t_i}^{(c)}}\left(\boldsymbol{\epsilon}_{t_i} - \boldsymbol{\Sigma}_\epsilon\partial_{x_{t_i}^{(c)}}\mathbf{g}_{t_i}^{(c)}\right) \tag{18}$$

In the case that $\mathbf{g}_{t_i}^{(c)}$ does not depend on $x_{t_i}^{(c)}$, the differentiation with respect to $x_{t_i}^{(c)}$ results to zero and therefore the local controls simplify to:

$$\mathbf{u}(\boldsymbol{\tau}_i) = \frac{\mathbf{R}^{-1}\mathbf{g}_{t_i}^{(c)}\mathbf{g}_{t_i}^{(c)T}}{\mathbf{g}_{t_i}^{(c)T}\mathbf{R}^{-1}\mathbf{g}_{t_i}^{(c)}}\boldsymbol{\epsilon}_{t_i} \tag{19}$$

[1]More precisely if $\mathbf{G}_{t_i}^{(c)} = \mathbf{G}^c$ then the term $\frac{\lambda}{2}\sum_{i=0}^{N-1}\log|\mathbf{h}_{t_j}|$ drops since it is state independent and it appears in both nominator and denominator in (12). In this case, the path cost is reduced to $\tilde{S}(\boldsymbol{\tau}_j) = S(\boldsymbol{\tau}_j)$.

Systems with square and state dependent control transition matrix:$\mathbf{u}(\boldsymbol{\tau}_i) = \boldsymbol{\epsilon}_{t_i} - \mathbf{G}_{t_i}^{(c)^{-1}} \mathbf{b}_{t_i}$. If the control transition matrix is not state dependent then we will have that:$\mathbf{u}(\boldsymbol{\tau}_i) = \boldsymbol{\epsilon}_{t_i}$. Previous work in (Kappen, 2005a, 2007, 2005b, Broek et al., 2008) are special cases of our generalized formulation. [2]

## 5 Parametrized Policies

Equipped with the theoretical framework of stochastic optimal control with path integrals, we can now turn to its application to reinforcement learning with parametrized policies. For this kind of direct policy learning, a general cost function $J = \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$ is usually assumed (Peters, 2007) and optimized over state $\mathbf{x}_t$ and action $\mathbf{a}_t$ trajectories $\boldsymbol{\tau} = (\mathbf{x}_{t_0}, \mathbf{a}_{t_o}, ..., \mathbf{x}_{t_N}, )$. Under the Markov property, the probability of a trajectory is $p(\boldsymbol{\tau}) = p(\mathbf{x}_{t_o}) \Pi_{i=1}^{N-1} p(\mathbf{x}_{t_{i+1}} | \mathbf{x}_{t_i}, \mathbf{a}_{t_i}) p(\mathbf{a}_{t_i} | \mathbf{x}_{t_i})$. As suggested in (Koeber and Peters, 2009), the mean of the stochastic policy $p(\mathbf{a}_{t_i} | \mathbf{x}_{t_i})$ is linearly parametrized as:

$$\mathbf{a}_{t_i} = \mathbf{g}_{t_i}^T (\boldsymbol{\theta} + \boldsymbol{\epsilon}_{t_i}) \qquad (20)$$

where $\mathbf{g}_{t_i}$ is a vector of basis functions and $\boldsymbol{\theta}$ is a parameter vector. For Gaussian noise $\boldsymbol{\epsilon}$ the policy distribution is $p(\mathbf{a}_{t_i} | \mathbf{x}_{t_i}) = N\left(\boldsymbol{\theta}^T \mathbf{g}_{t_i}, \boldsymbol{\Sigma}_{t_i}\right)$. In our work, we use a special case of parametrized policies in form of Dynamic Movement Primitives (DMPs) (Ijspeert et al., 2003), which are expressed as:

$$\begin{aligned} \frac{1}{\tau} \dot{z}_t &= f_t + \mathbf{g}_t^T (\boldsymbol{\theta} + \boldsymbol{\epsilon}_t), \qquad (21) \\ \frac{1}{\tau} \dot{y}_t &= z_t, \quad \frac{1}{\tau} \dot{x}_t = -\alpha x_t \end{aligned}$$

with $f_t = \alpha_z (\beta_z (g - y_t) - z_t)$. These policies code a learnable point attractor for a movement from $y_{t_0}$ to the goal $g$, where $\theta$ determines the shape of the attractor – for more details and the definition of the basis functions $\mathbf{g}_t$ see (Ijspeert et al., 2003). The DMP equations are obviously of the form of our control system (3), just with a row vector as control transition matrix $\mathbf{G}_t^{(c)} = \mathbf{g}_t{}^T \in \Re^{1 \times p}$. Thus, we can treat the parameters $\theta$ as if they were control commands, and, after some algebra and simplifications, we derive the **P**olicy **I**mprovement with **P**ath **I**ntegrals (**PI**$^2$) which is summarized in table 1.

The **PI**$^2$ is an iterative algorithm. As it is illustrated in Table 1, at every iteration we create K sample paths

Table 1: Pseudocode of the **PI**$^2$ algorithm for a 1D Parameterized Policy (Note that the discrete time step $dt$ was absorbed as a constant multiplier in the cost terms). In **step 4**, the parameter vector is updated coefficient wise, using $m$ to denote each of the $p$ coefficients. The weights $w_{p,t_i}$ in this update come from the basis functions representation that is used in the nonlinear function $f_t$ of the DMPs in (22) (cf. (Ijspeert et al., 2003)).

```
• Given:
    – An immediate cost function $r_t = q_t + \boldsymbol{\theta}_t^T \mathbf{R} \boldsymbol{\theta}_t$
    – A terminal cost term $\phi_{t_N}$
    – A stochastic parameterized policy $\mathbf{a}_t = \mathbf{g}_t^T (\boldsymbol{\theta} + \boldsymbol{\epsilon}_t)$ (cf. 20)
    – The basis function $\mathbf{g}_{t_i}$ from the system dynamics
    – The variance $\Sigma_\epsilon$ of the mean-zero noise $\boldsymbol{\epsilon}_t$
    – The initial parameter vector $\boldsymbol{\theta}$
• Repeat until convergence of the trajectory cost $R$:
    – step 1: Create $K$ roll-outs of the system from the same start state $\mathbf{x}_0$ using stochastic parameters $\boldsymbol{\theta} + \boldsymbol{\epsilon}_t$ at every time step
    – step 2: For all $K$ roll-outs, compute:
        * step 2.1: $\mathbf{M}_{t_j,k} = \frac{\mathbf{R}^{-1} \mathbf{g}_{t_j,k} \, \mathbf{g}_{t_j,k}^T}{\mathbf{g}_{t_j,k}^T \mathbf{R}^{-1} \mathbf{g}_{t_j,k}}$
        * step 2.2: Compute the cost for each sampled trajectory: $S(\boldsymbol{\tau}_{i,k}) = \phi_{t_N,k} + \sum_{j=i}^{N-1} q_{t_j,k} + \frac{1}{2} \sum_{j=i+1}^{N-1} (\boldsymbol{\theta} + \mathbf{M}_{t_j,k} \boldsymbol{\epsilon}_{t_j,k})^T \mathbf{R} (\boldsymbol{\theta} + \mathbf{M}_{t_j,k} \boldsymbol{\epsilon}_{t_j,k})$
        * step 2.3: $P(\boldsymbol{\tau}_{i,k}) = \frac{e^{-\frac{1}{\lambda} S(\boldsymbol{\tau}_{i,k})}}{\sum_{k=1}^{K} [e^{-\frac{1}{\lambda} S(\boldsymbol{\tau}_{i,k})}]}$
    – step 3: For all $i$ time steps, compute:
        * step 3.1: $\delta\boldsymbol{\theta}_{t_i} = \sum_{k=1}^{K} [P(\boldsymbol{\tau}_{i,k}) \mathbf{M}_{t_i,k} \, \boldsymbol{\epsilon}_{t_i,k}]$
    – step 4: Compute $\delta\theta^m = \frac{\sum_{i=0}^{N-1} (N-i) w_{t_i}^m \delta\theta_{t_i}^m}{\sum_{i=0}^{N-1} (N-i) w_{t_i}^m}$
    – step 5: Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}$
    – step 6: Create one noiseless roll-out to check the trajectory cost $R = \phi_{t_N} + \sum_{i=0}^{N-1} r_{t_i}$
```

starting from the initial state and ending in the target state. To create these rollouts we propagate the stochastic dynamics forward given an initial control (parameter) vector. Given the sample paths we are calculating the correction in controls $\delta\boldsymbol{\theta}$. For the case of an horizon of length T with discretization step $dt$ and K rollouts there are $T/dt$ different $\delta\boldsymbol{\theta}_{t_i}$ vectors of dimensionality $\delta\boldsymbol{\theta}_{t_i} \in \Re^{K \times 1}$. Each $\delta\boldsymbol{\theta}_{t_i}$ is calculated in step (3.1). The main intuition here is based on the observation that $\epsilon_{t_{i,k}}$ is the sampled control variation at time $t_i$ that was used to create the k-th sample path and $S(\boldsymbol{\tau}_{i,k})$ in step (2.2) is the cost of the resulting k-th sample path. At every time $t_i$ there are

[2]In fact, for stochastic systems with state independent control transition matrix $\mathbf{G}_{t_o}^{(c)} = \mathbf{G}^{(c)}$ the term $\mathbf{b}_{t_o} = 0_{l \times 1}$ since $\mathbf{h}_{t_o}$ becomes state independent and therefore $\partial_{\mathbf{x}_{t_o}^{(cj)}} \mathbf{h}_{t_o} = 0$. In such case the local controls are reduced to $\mathbf{u}(\boldsymbol{\tau}_o) = \boldsymbol{\epsilon}_{t_0}$.

K different sampled controllers $\epsilon_{t_{i,k}}$ that correspond to the K rollouts. The correction term $\delta\boldsymbol{\theta}_{t_i}$ is finally calculated by a weighted average of the sampled controllers (spatial averaging). The weights are function of the corresponding cost $S(\boldsymbol{\tau}_{i,k})$. Paths with high cost have low weight or probability, while paths with low cost have high probability and therefore will more strongly affect the correction term $\delta\boldsymbol{\theta}_{t_i}$. This averaging process is repeated for all time instances of the time horizon. Since we are performing sampling in an iterative way, the resulting control corrections $\delta\boldsymbol{\theta}_{t_i}$ are averaged over the time horizon step (4) (time averaging). The resulting control correction $\delta\boldsymbol{\theta}$ is used for the final update $\boldsymbol{\theta}^{(new)} = \boldsymbol{\theta}^{(old)} + \delta\boldsymbol{\theta}$ in step (5). The last two operations of time averaging and parameter update, are motivated from the need to avoid the curse of dimensionality which results from batch sampling of the entire state space. The entire $\mathbf{PI}^2$ formulation allows an incremental updating of $\boldsymbol{\theta}$. The new parameter vector $\boldsymbol{\theta}_{new}$, with the addition of Gaussian noise $\boldsymbol{\epsilon}$, corresponds to the control that is used to create the K sample paths in the next iteration.

The parameter $\lambda$ regulates the sensitivity of the exponentiated cost and can automatically be optimized for every time step $i$ to maximally discriminate between the experienced trajectories. Moreover, a constant term can be subtracted from the cost $S(\boldsymbol{\tau}_i)$ as long as all $S(\boldsymbol{\tau}_i)$ is positive. Thus, for a given number of roll-outs, we compute the exponential term in (12) as

$$e^{\frac{1}{\lambda}S(\boldsymbol{\tau}_i)} = \exp\left(-c\frac{S(\boldsymbol{\tau}_i) - \min S(\boldsymbol{\tau}_i)}{\max S(\boldsymbol{\tau}_i) - \min S(\boldsymbol{\tau}_i)}\right) \quad (22)$$

with $c = 10$ in all our evaluations – this procedure eliminates $\lambda$ and leaves the variance of the exploration noise $\epsilon$ as the only open parameter for $\mathbf{PI}^2$. It should be noted that the equations for $\mathbf{PI}^2$ have no numerical pitfalls: no matrix inversions and no learning rates [3], rendering $\mathbf{PI}^2$ to be very easy to use.

## 6 Evaluations

We evaluated $\mathbf{PI}^2$ in several synthetic examples in comparison with REINFORCE, GPOMDP, eNAC, and, when possible, PoWER. Except for PoWER, all algorithms are suitable for optimizing immediate reward functions of the kind $r_t = q_t + \mathbf{u}_t\mathbf{R}\mathbf{u}_t$. PoWER requires that the immediate reward behaves like an improper probability. This property is incompatible with $r_t = q_t + \mathbf{u}_t\mathbf{R}\mathbf{u}_t$ and requires some special nonlinear transformations, which usually change the nature of the optimization problem, such that PoWER optimizes a different cost function. Thus, only one of the examples below has a compatible cost function for all algorithms, including PoWER. In all examples below, exploration noise and, when applicable, learning rates, were tuned for every individual algorithms to achieve the best possible numerically stable performance. Exploration noise was only added to the maximally activated basis function in a motor primitive[4], and the noise was kept constant for the entire time that this basis function had the highest activation – empirically, this trick helped improving the learning speed of all algorithms.

### 6.1 Learning Optimal Performance of a 1 DOF Reaching Task

The first evaluation considers learning optimal parameters for a 1 DOF DMP (cf. Equation 22). The immediate cost and terminal cost are, respectively:

$$\begin{aligned} r_t &= 0.5f_t^2 + 5000\ \boldsymbol{\theta}^T\boldsymbol{\theta} \\ \phi_{t_N} &= 10000(\dot{y}_{t_N}^2 + 10(g - y_{t_N})^2) \end{aligned}$$

with $y_{t_0} = 0$ and $g = 1$ – we use *radians* as units motivated by our interest in robotics application, but we could also avoid units entirely. The interpretation of this cost is that we would like to reach the goal $g$ with high accuracy while minimizing the acceleration of the movement and while keeping the parameter vector short. Each algorithm was run for 15 trials to compute a parameter update, and a total of 1000 updates were performed. Note that 15 trials per update were chosen as the DMP had 10 basis functions, and the eNAC requires at least 11 trials to perform a numerically stable update due to its matrix inversion. The motor primitives were initialized to approximate a 5th order polynomial as point-to-point movement (cf. Figure 1a,b), called a minimum-jerk trajectory in the motor control literature; the movement duration was 0.5 seconds, which is similar to normal human reaching movements. Gaussian noise of $N(0, 0.1)$ was added to the initial parameters of the movement primitives in order to have different initial conditions for every run of the algorithms. The results are given in Figure 1. Figure 1a,b show the initial (before learning) trajectory generated by the DMP together with the learning results of the four different algorithms after learning – essentially, all algorithms achieve the same result such that all trajectories lie on top of each other. In Figure 1c, however, it can be seen that $\mathbf{PI}^2$ outperforms the gradient algorithms by an order of magnitude.

Figure 1d illustrates learning curves for the same task as in Figure 1c, just that parameter updates are computed already after two roll-outs – the eNAC was ex-

[3] $\mathbf{R}$ is a user design parameter and usually chosen to be diagonal and invertible.

[4] I.e., the noise vector in (20) has only one non-zero component.

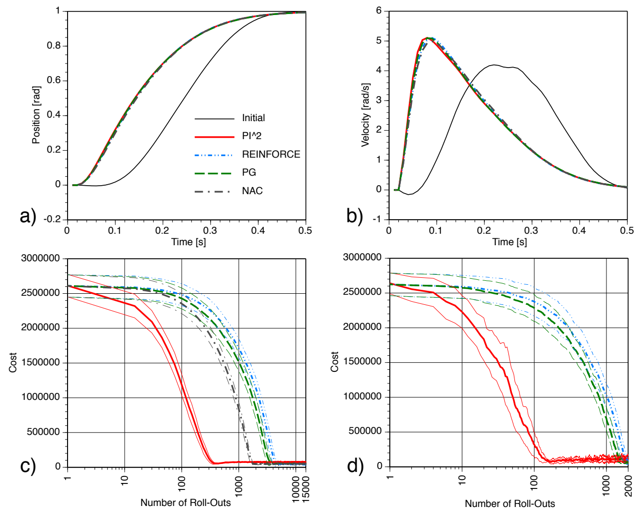

Figure 1: Comparison of reinforcement learning of an optimized movement with motor primitives. a) Position trajectories of the initial trajectory (before learning) and the results of all algorithms after learning – the different algorithms are essentially non distinguishable. b) The same as a), just using the velocity trajectories. c) Average learning curves for the different algorithms with 1 std error bars from averaging 10 runs for each of the algorithms. d) Learning curves for the different algorithms when only two roll-outs are used per update (note that the eNAC cannot work in this case and is omitted).

cluded from this evaluation as it would be too heuristic to stabilize its ill-conditioned matrix inversion that results from such few roll-outs. **PI**$^2$ continues to converge much faster than the other algorithms even in this special scenario. However, there are some noticeable fluctuation after convergence. This noise around the convergence baseline is caused by using only two noisy roll-outs to continue updating the parameters, which causes continuous parameter fluctuations around the optimal parameters. Annealing the exploration noise, or just adding the optimal trajectory from the previous parameter update as one of the roll-outs for the next parameter update can alleviate this issue – we do not illustrate such little "tricks" in this paper as they really only affect fine tuning of the algorithm.

## 6.2 Learning Optimal Performance of a 1 DOF Via-Point Task

The second evaluation was identical to the first evaluation, just that the cost function now forced the movement to pass through an intermediate via-point at $t = 300ms$. This evaluation is an abstract approximation of hitting a target, e.g., as in playing tennis, and

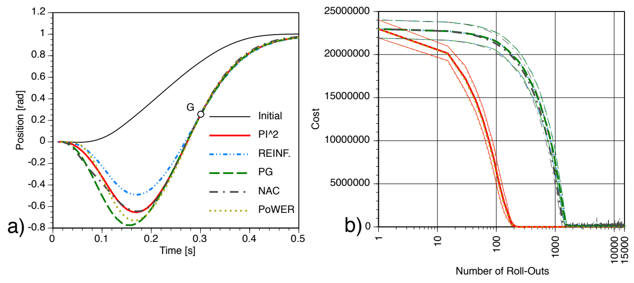

Figure 2: Comparison of reinforcement learning of an optimized movement with motor primitives for passing through an intermediate target $G$. a) Position trajectories of the initial trajectory (before learning) and the results of all algorithms after learning. b) Average learning curves for the different algorithms with 1 std error bars from averaging 10 runs for each of the algorithms.

requires a significant change in how the movement is performed relative to the initial trajectory (Figure 2a). The cost function was

$$r_{300ms} = 100000000(G - y_{t_{300ms}})^2 \qquad \phi_{t_N} = 0$$

with $G = 0.25$. Only this single reward was given. For this cost function, the PoWER algorithm can be applied, too, with cost function $\tilde{r}_{300ms} = exp(-1/\lambda \quad r_{300ms})$ and $\tilde{r}_{t_i} = 0$ otherwise. This transformed cost function has the same optimum as $r_{300ms}$. The resulting learning curves are given in Figure 2 and resemble the previous evaluation: **PI**$^2$ outperforms the gradient algorithms by roughly an order of magnitude, while all the gradient algorithms have almost identical learning curves. As was expected from the similarity of the update equations, PoWER and **PI**$^2$ have in this special case the same performance and are hardly distinguishable in Figure 2. Figure 2a demonstrates that all algorithms pass through the desired target $G$, but that there are remaining differences between the algorithms in how they approach the target $G$ – these difference have a small numerical effect in the final cost (where **PI**$^2$ and PoWER have the lowest cost), but these difference are hardly task relevant.

## 6.3 Learning Optimal Performance of a Multi-DOF Via-Point Task

A third evaluation examined the scalability of our algorithms to a high-dimensional and highly redundant learning problem. Again, the learning task was to pass through an intermediate target $G$, just that a $d = 2, 10$, or $50$ dimensional motor primitive was employed. We assume that the multi-DOF systems model planar robot arms, where $d$ links of equal length $l = 1/d$ are connected in an open chain with revolute joints. Essentially, these robots look like a multi-segment snake in a plane, where the tail of the snake is fixed at the origin of the 2D coordinate system, and the head of the snake can be moved in the 2D plane by changing the joint angles between all the links. Figure 3b,d,f illustrate the movement over time of these robots: the initial position of the robots is when all joint angles are zero and the robot arm completely coincides with the $x$-axis of the coordinate frame. The goal states of the motor primitives command each DOF to move to a joint angle, such that the entire robot configuration afterwards looks like a semi-circle where the most distal link of the robot (the endeffector) touches the $y$-axis. The higher priority task, however, is to move the endeffector through a via-point $G = (0.5, 0.5)$. To formalize this task as a reinforcement learning problem, we denote the joint angles of the robots as $\xi_i$, with $i = 1, 2, ..., d$, such that the first line of (22) reads now as $\ddot{\xi}_{i,t} = f_{i,t} + \mathbf{g}_{i,t}^T(\boldsymbol{\theta}_i + \boldsymbol{\epsilon}_{i,t})$ – this small change of notation is to avoid a clash of variables with the $(x.y)$ task space of the robot. The endeffector position is computed as:

$$x_t = \frac{1}{d}\sum_{i=1}^{d} cos(\sum_{j=1}^{i} \xi_{j,t})$$

$$y_t = \frac{1}{d}\sum_{i=1}^{d} sin(\sum_{j=1}^{i} \xi_{j,t})$$

The immediate reward function for this problem is defined as

$$\begin{aligned} r_t &= \frac{\sum_{i=1}^{d}(d+1-i)\left(0.1 f_{i,t}^2 + 0.5\ \boldsymbol{\theta}_i^T\boldsymbol{\theta}\right)}{\sum_{i=1}^{d}(d+1-i)} \quad (23) \\ \Delta r_{300ms} &= 1e8\left((0.5 - x_{t_{300ms}})^2 + (0.5 - y_{t_{300ms}})^2\right) \\ \phi_{t_N} &= 0 \end{aligned}$$

where $\Delta r_{300ms}$ is added to $r_t$ at time $t = 300ms$, i.e., we would like to pass through the via-point at this time. The individual DOFs of the motor primitive were initialized as in the 1 DOF examples above. The cost term in (23) penalizes each DOF for using high accelerations and large parameter vectors, which is a critical component to achieve a good resolution of redundancy in the arm. Equation (23) also has a weighting term $d+1-i$ that penalizes DOFs proximal to the origin more than those that are distal to the origin — intuitively, applied to human arm movements, this would mean that wrist movements are cheaper than shoulder movements, which is motivated by the fact that the wrist has much lower mass and inertia and is thus energetically more efficient to move.

| Algorithm | 2-DOFs | 10-DOFs | 50-DOFs |
|---|---|---|---|
| **PI**$^2$ | $98000 \pm 5000$ | $15700 \pm 1300$ | $2800 \pm 150$ |
| REINFORCE | $125000 \pm 2000$ | $22000 \pm 700$ | $19500 \pm 24000$ |
| PG | $128000 \pm 2000$ | $28000 \pm 23000$ | $27000 \pm 40000$ |
| NAC | $113000 \pm 10000$ | $48000 \pm 8000$ | $22000 \pm 2000$ |

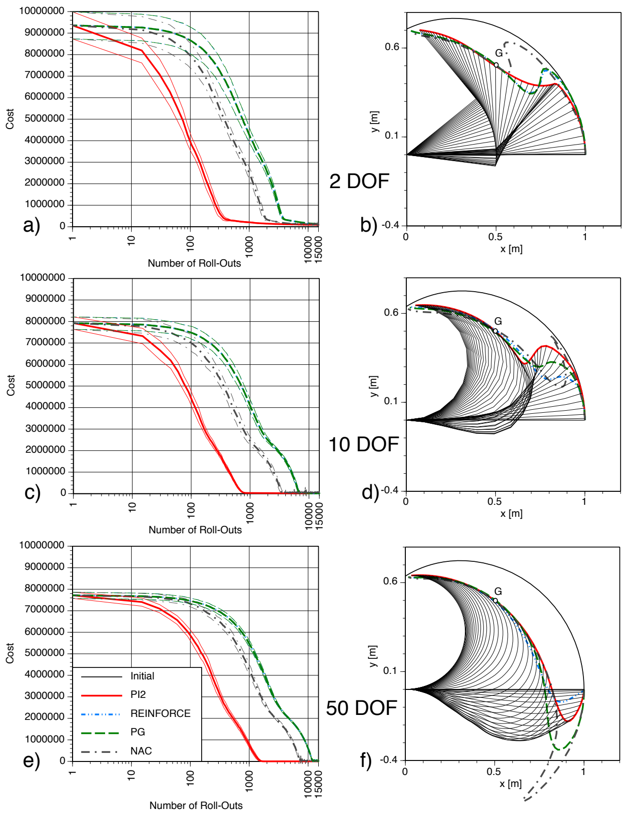

Figure 3: Comparison of learning multi-DOF movements (2,10, and 50 DOFs) with planar robot arms passing through a via-point $G$. a,c,e) illustrate the learning curves for different RL algorithms, while b,d,f) illustrate the endeffector movement after learning for all algorithms. Additionally, b,d,f) also show the initial endeffector movement, before learning to pass through $G$, and a "stroboscopic" visualization of the arm movement for the final result of **PI**$^2$ (the movements proceed in time starting at the very right and ending by (almost) touching the $y$ axis).

The results of this experiment are summarized in Figure 3. The learning curves in the left column demonstrate again that **PI**$^2$ has an order of magnitude faster learning performance than the other algorithms, irrespective of the dimensionality. **PI**$^2$ also converges to the lowest cost in all examples:

Figure 3 also illustrates the path taken by the endeffector before and after learning. All algorithms manage to pass through the via-point $G$ appropriately, although the path particularly before reaching the via-point can be quite different across the algorithms. Given that

$\mathbf{PI}^2$ reached the lowest cost with low variance in all examples, it appears to have found the best solution. We also added a "stroboscopic" sketch of the robot arm for the $\mathbf{PI}^2$ solution, which proceeds from the very right to the left as a function of time. It should be emaphasized that there was absolutely no parameter tuning needed to achieve the $\mathbf{PI}^2$ results, while all gradient algorithms required readjusting of learning rates for every example to achieve best performance.

## 7 Discussion

We introduced Policy Improvement with Path Integrals ($\mathbf{PI}^2$) as a novel algorithm for learning a parametrized policy in reinforcement learning. $\mathbf{PI}^2$ has a very sound foundation in first order principles of stochastic optimal control. It is a probabilistic learning method without open tuning parameters, except for the exploration noise. In our evaluations, $\mathbf{PI}^2$ outperformed gradient algorithms significantly. It is also numerically simpler and has easier cost function design than previous probabilistic RL methods that require that immediate rewards are pseudo-probabilities. The similarity of $\mathbf{PI}^2$ with algorithms based on probability matching indicates that the principle of probability matching seems to approximate a stochastic optimal control framework. $\mathbf{PI}^2$ makes a strong link to the dynamic system to be optimized. Complete knowledge of the control system allows a model-based approach, while partial knowledge of the control transition matrix could allow a semi-model based approach. Our learning of motor primitives was essentially a model-free application of path-integrals, where only the parametric form of the control policy was given. Our simulated robot learning example demonstrated that $\mathbf{PI}^2$ can scale to high dimensional control systems.

## 8 Acknowledgement

This research was supported in part by National Science Foundation grants ECS-0325383, IIS-0312802, IIS-0082995, IIS-9988642, ECS-0326095, the DARPA program on Learning Locomotion, and the ATR Computational Neuroscience Laboratories. J.B. holds a prospective researcher fellowship from the Swiss National Science Foundation. E.T. was supported by a Myronis Fellowship.

## References

Broek, B. v. d., Wiegerinck, W., and Kappen, B. (2008). Graphical model inference in optimal control of stochastic multi-agent systems. *Journal of Artificial Intelligence Research*, 32(1):95–122.

Dayan, P. and Hinton, G. (1997). Using em for reinforcement learning. *Neural Computation*, 9.

Fleming, W. H. and Soner, H. M. (2006). *Controlled Markov processes and viscosity solutions*. Applications of mathematics. Springer, New York, 2nd edition.

Ijspeert, A., Nakanishi, J., and Schaal, S. (2003). Learning attractor landscapes for learning motor primitives. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 1547–1554. Cambridge, MA: MIT Press. clmc.

Kappen, H. J. (2005a). Linear theory for control of nonlinear stochastic systems. *Phys Rev Lett*, 95(20):200201. Journal Article United States.

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

Kappen, H. J. (2007). An introduction to stochastic control theory, path integrals and reinforcement learning. In Marro, J., Garrido, P. L., and Torres, J. J., editors, *Cooperative Behavior in Neural Systems*, volume 887 of *American Institute of Physics Conference Series*, pages 149–181.

Koeber, J. and Peters, J. (2009). Learning motor primitives in robotics. In Schuurmans, D., Benigio, J., and Koller, D., editors, *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, Vancouver, BC, Dec. 8-11. Cambridge, MA: MIT Press. clmc.

Oksendal, B. K. (2003). *Stochastic differential equations : an introduction with applications*. Universitext. Springer, Berlin ; New York, 6th edition.

Peters, J. (2007). *Machine learning of motor skills for robotics*. PhD thesis.

Peters, J. and Schaal, S. (2008a). Learning to control in operational space. *International Journal of Robotics Research*, 27:197–212. clmc.

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

Stengel, R. F. (1994). *Optimal control and estimation*. Dover books on advanced mathematics. Dover Publications, New York.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning : An introduction*. Adaptive computation and machine learning. MIT Press, Cambridge.

Toussaint, M. and Storkey, A. (2006). Probabilistic inference for solving discrete and continuous state markov decision processes. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 945–952.

Yong, J. (1997). Relations among odes, pdes, fsdes, bsdes, and fbsdes. volume 3, pages 2779–2784.