Foundations and Trends® in
Robotics
Vol. 2, No 1–2 (2013) 1–141
© 2013 Deisenroth, Neumann, Peters
DOI: 10.1561/2300000021

A Survey on Policy Search for Robotics

Marc Peter Deisenroth*[1], Gerhard Neumann*[2] and Jan Peters[3]

[1] *Technische Universität Darmstadt, Germany*
Imperial College London, UK
[2] *Technische Universität Darmstadt, Germany*
[3] *Technische Universität Darmstadt, Germany*
Max Planck Institute for Intelligent Systems, Germany

Abstract

Policy search is a subfield in reinforcement learning which focuses on finding good parameters for a given policy parametrization. It is well suited for robotics as it can cope with high-dimensional state and action spaces, one of the main challenges in robot learning. We review recent successes of both model-free and model-based policy search in robot learning.
Model-free policy search is a general approach to learn policies based on sampled trajectories. We classify model-free methods based on their policy evaluation strategy, policy update strategy and exploration strategy and present an unified view on existing algorithms. Learning a policy is often easier than learning an accurate forward model, and, hence, model-free methods are more frequently used in practice. However, for each sampled trajectory, it is necessary to interact with the robot, which can be time consuming and challenging in practice.

* Both authors contributed equally.

Model-based policy search addresses this problem by first learning a simulator of the robot's dynamics from data. Subsequently, the simulator generates trajectories that are used for policy learning. For both model-free and model-based policy search methods, we review their respective properties and their applicability to robotic systems.

Contents

1 Introduction

From simple house-cleaning robots to robotic wheelchairs and general transport robots the number and variety of robots used in our everyday life are rapidly increasing. To date, the controllers for these robots are largely designed and tuned by a human engineer. Programming robots is a tedious task that requires years of experience and a high degree of expertise. The resulting programmed controllers are based on assuming exact models of both the robot's behavior and its environment. Consequently, hard-coding controllers for robots has its limitations when a robot has to adapt to new situations or when the robot/environment cannot be modeled sufficiently accurately. Hence, there is a gap between the robots currently used and the vision of incorporating fully autonomous robots. In *robot learning*, machine learning methods are used to automatically extract relevant information from data to solve a robotic task. Using the power and flexibility of modern machine learning techniques, the field of robot control can be further automated, and the gap toward autonomous robots, e.g., for general assistance in households, elderly care, and public services can be narrowed substantially.

1.1 Robot Control as a Reinforcement Learning Problem

In most tasks, robots operate in a high-dimensional state space $\boldsymbol{x}$ composed of both internal states (e.g., joint angles, joint velocities, end-effector pose, and body position/orientation) and external states (e.g., object locations, wind conditions, or other robots). The robot selects its motor commands $\boldsymbol{u}$ according to a control policy π. The control policy can either be stochastic, denoted by $\pi(\boldsymbol{u}|\boldsymbol{x})$, or deterministic, which we will denote as $\boldsymbol{u} = \pi(\boldsymbol{x})$. The motor commands $\boldsymbol{u}$ alter the state of the robot and its environment according to the probabilistic transition function $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$. Jointly, the states and actions of the robot form a *trajectory* $\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \dots)$, which is often also called a *roll-out* or a *path*.

We assume that a numeric scoring system evaluates the performance of the robot system during a task and returns an accumulated reward signal $R(\boldsymbol{\tau})$ for the quality of the robot's trajectory. For example, the reward $R(\boldsymbol{\tau})$ may include a positive reward for a task achievement and negative rewards, i.e., costs, that punish energy consumption. Many of the considered motor tasks are stroke-based movements, such as returning a tennis ball or throwing darts. We will refer to such tasks as *episodic learning tasks* as the execution of the task, the *episode*, ends after a given number T of time steps. Typically, the accumulated reward $R(\boldsymbol{\tau})$ for a trajectory is given as

$$R(\boldsymbol{\tau}) = r_T(\boldsymbol{x}_T) + \sum_{t=0}^{T-1} r_t(\boldsymbol{x}_t, \boldsymbol{u}_t)\,, \tag{1.1}$$

where r_t is an instantaneous reward function, which might be a punishment term for the consumed energy, and r_T is a final reward, such as quadratic punishment term for the deviation to a desired goal posture. For many episodic motor tasks the policy is modeled as time-dependent policy, i.e., either a stochastic policy $\pi(\boldsymbol{u}_t|\boldsymbol{x}_t, t)$ or a deterministic policy $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t, t)$ is used.

In some cases, the infinite-horizon case is considered

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{x}_t, \boldsymbol{u}_t)\,, \tag{1.2}$$

where $\gamma \in [0, 1)$ is a discount factor that discounts rewards further in the future.

Many tasks in robotics can be phrased as choosing a (locally) optimal control policy π^* that maximizes the expected accumulated reward

$$J_\pi = \mathbb{E}[R(\boldsymbol{\tau})|\pi] = \int R(\boldsymbol{\tau}) p_\pi(\boldsymbol{\tau}) d\boldsymbol{\tau}\,, \tag{1.3}$$

where $R(\boldsymbol{\tau})$ defines the objectives of the task, and $p_\pi(\boldsymbol{\tau})$ is the distribution over trajectories $\boldsymbol{\tau}$. For a stochastic policy $\pi(\boldsymbol{u}_t|\boldsymbol{x}_t, t)$, the trajectory distribution is given as

$$p_\pi(\boldsymbol{\tau}) = p(\boldsymbol{x}_0) \prod_{t=0}^{T-1} p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)\pi(\boldsymbol{u}_t|\boldsymbol{x}_t, t), \tag{1.4}$$

where $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ is given by the system dynamics of the robot and its environment. For a deterministic policy, $p_\pi(\boldsymbol{\tau})$ is given as

$$p_\pi(\boldsymbol{\tau}) = p(\boldsymbol{x}_0) \prod_{t=0}^{T-1} p\left(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \pi(\boldsymbol{x}_t, t)\right). \tag{1.5}$$

With this general reinforcement learning (RL) problem set-up, many tasks in robotics can be naturally formulated as *Reinforcement Learning* (RL) problems. However, robot RL poses three main challenges, which have to be solved: The RL algorithm has to manage (i) high-dimensional continuous state and action spaces, (ii) strong real-time requirements, and (iii) the high costs of robot interactions with its environment.

Traditional methods in RL, such as TD-learning [80], typically try to estimate the expected long-term reward of a policy for each state $\boldsymbol{x}$ and time step t, also called the *value function* $V_t^\pi(\boldsymbol{x})$. The value function is used to calculate the quality of an executing action $\boldsymbol{u}$ in state $\boldsymbol{x}$. This quality assessment is subsequently utilized to directly compute the policy by action selection or to update the policy π. However, value function methods struggle with the challenges encountered in robot RL, as these approaches require filling the complete state-action space with data. In addition, the value function is computed iteratively by the use of bootstrapping, which often results in a bias in the quality assessment of the state action pairs if we need to resort to value function approximation techniques as it is the case for continuous state spaces. Consequently, value function approximation turns out to be a very difficult problem in high-dimensional state and action spaces. Another major issue is that value functions are often discontinuous, especially when the non-myopic policy differs from a myopic policy. For instance, the value function of the under-powered pendulum swing-up is discontinuous along the manifold where the applicable torque is just not sufficient to swing the pendulum up [22]. Any error in the value function will eventually propagate through to the policy.

In a classical RL set-up, we seek a policy without too specific prior information. Key to successful learning is the exploration strategy of the learner to discover rewarding states and trajectories. In a robotics context, arbitrary exploration is not desired if not discouraged since the robot can easily be damaged. Therefore, the classical RL paradigm in a robotics context is not directly applicable since exploration needs to take hardware constraints into account. Two ways of implementing cautious exploration are to either avoid significant changes in the policy [57] or to explicitly discourage entering undesired regions in the state space [21].

In contrast to value-based methods, *Policy Search* (PS) methods use parametrized policies $\pi_{\boldsymbol{\theta}}$. They directly operate in the parameter space $\boldsymbol{\Theta}$, $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, of parametrized policies, and typically avoid learning a value function. Many methods do so by directly using the experienced reward to come from the rollouts as quality assessment for state action pairs instead of using the rather dangerous bootstrapping used in value-function approximation. The usage of parametrized policies allows for scaling RL into high dimensional continuous action spaces by reducing the search space of possible policies.

Policy search allows task-appropriate pre-structured policies, such as movement primitives [71], to be integrated straightforwardly. Additionally, imitation learning from an expert's demonstrations can be used to obtain an initial estimate for the policy parameters [58]. Finally, by selecting a suitable policy parametrization, stability and robustness guarantees can be given [11]. All these properties simplify the robot learning problem and permit the successful application of reinforcement learning to robotics. Therefore, PS is often the RL approach of choice in robotics since it is better at coping with the inherent challenges of robot reinforcement learning. Over the last decade, a series of fast policy search algorithms have been proposed and shown to work well on real systems [38, 53, 58, 86, 7, 21, 17]. In this review, we provide a general overview, summarize the main concepts behind current policy search approaches, and discuss relevant robot applications of these policy search methods. We focus mainly on those aspects of RL that are predominant for robot learning, i.e., learning in high-dimensional continuous state and action spaces and a high data-efficiency and local exploration. Other important aspects of RL, such as the exploration-exploitation trade-off, feature selection, using structured models or value function approximation are not covered in this monograph.

1.2 Policy Search Taxonomy

Numerous policy search methods have been proposed in the last decade, and several of them have been used successfully in the domain of robotics. In this monograph, we review several important recent developments in policy search for robotics. We distinguish between model-free policy search methods (Section 2), which learn policies directly based on sampled trajectories, and model-based approaches (Section 3), which use the sampled trajectories to first build a model of the state dynamics, and, subsequently, use this model for policy improvement.

Figure 1.1 categorizes policy search into model-free policy search and model-based policy search and distinguishes between different policy update strategies. The policy updates in both model-free and model-based policy search (green blocks) are based on either policy gradients (PG), expectation maximization (EM)-based updates, or information-theoretic insights (Inf.Th.). While all three update strategies are fairly well explored in model-free policy search, model-based policy search almost exclusively focuses on PG to update the policy.

Model-free policy search uses stochastic trajectory generation, i.e., the trajectories are generated by “sampling” from the robot $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ and the policy $\pi_{\boldsymbol{\theta}}$. This means, a system model is not

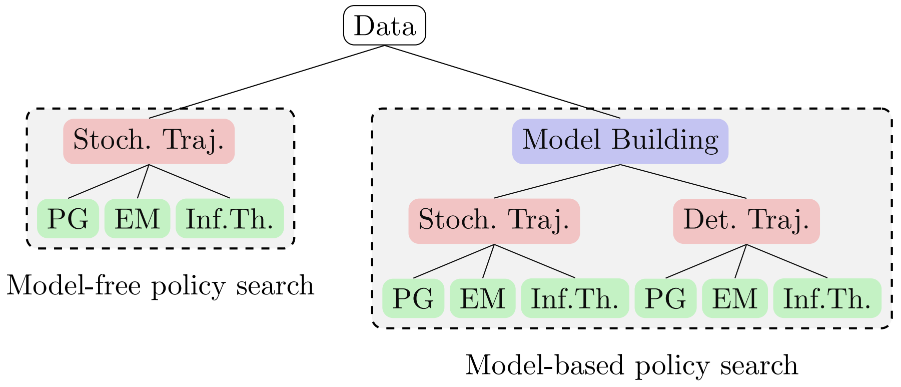


Fig. 1.1 Categorization of policy search into model-free policy search and model-based policy search. In the model-based case (right sub-tree), data from the robot is used to learn a model of the robot (blue box). This model is then used to generate trajectories. Here, we distinguish between stochastic trajectory generation and deterministic trajectory prediction. Model-free policy search (left sub-tree) uses data from the robot directly as a trajectory for updating the policy. The policy updates in both model-free and model-based policy search (green blocks) are based on either policy gradients (PG), expectation maximization (EM)-based updates, or information-theoretic insights (Inf.Th.).

explicitly required; We just have to be able to sample trajectories from the real robot. In the model-based case (right sub-tree), we can either use stochastic trajectory generation or deterministic trajectory prediction. In the case of stochastic trajectory generation, the learned models are used as simulator for sampling trajectories. Hence, learned models can easily be combined with model-free policy search approaches by exchanging the "robot" with the learned model of the robot's dynamics. Deterministic trajectory prediction does not sample trajectories, but analytically predicts the trajectory distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$. Typically, deterministic trajectory prediction is computationally more involved than sampling trajectories from the system. However, for the subsequent policy update, deterministic trajectory prediction can allow for analytic computation of gradients, which can be advantageous over stochastic trajectory generation, where these gradients can only be approximated.

1.2.1 Model-free and Model-based Policy Search

Model-free policy search methods use real robot interactions to create sample trajectories $\boldsymbol{\tau}^{[i]}$. While sampling trajectories is relatively straightforward in computer simulation, when working with robots, the generation of each "sample" typically needs some level of human supervision. Consequently, trajectory generation with the real system is considerably more time consuming than working with simulated systems. Furthermore, real robot interactions causes wear and tear in non-industrial robots. However, in spite of the relatively high number of required robot interactions for model-free policy search, learning a policy is often easier than learning accurate forward models, and, hence, model-free policy search is more widely used than model-based methods.

Model-based policy search methods attempt to address the problem of sample inefficiency by using the observed trajectories $\boldsymbol{\tau}^{[i]}$ to learn a forward model of the robot's dynamics and its environment. Subsequently, this forward model is used for *internal* simulations of the robot's dynamics and environment, based on which the policy is learned. Model-based PS methods have the potential to require fewer interactions with the robot and to efficiently generalize to unforeseen situations [6]. While the idea of using models in the context of robot learning is well-known since the 1980s [2], it has been limited by its strong dependency on the quality of the learned models. In practice, the learned model is *not* exact, but only a more or less accurate approximation to the real dynamics. Since the learned policy is inherently based on internal simulations with the learned model, inaccurate models can, therefore, lead to control strategies that are not robust to model errors. In some cases, learned models may be physically implausible and contain negative masses or negative friction coefficients. These implausible effects are often exploited by the policy search algorithm, resulting in a poor quality of the learned policy. This effect can be alleviated by using models that explicitly account for model errors [72, 20]. We will discuss such methods in Section 3.

1.3 Typical Policy Representations

Typical policy representations, which are used for policy search can be categorized into time-independent representations $\pi(\boldsymbol{x})$ and time-dependent representations $\pi(\boldsymbol{x}, t)$. Time-independent representations use the same policy for all time steps, and, hence, often require a complex parametrization. Time-dependent representations can use different policies for different time steps, allowing for a potentially simpler structure of the individual policies can be used.

We will describe all policy representations in their deterministic formulation $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$. In stochastic formulations, typically a zero-mean Gaussian noise vector $\boldsymbol{\epsilon}_t$ is added to $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$. In this case, the parameter vector $\boldsymbol{\theta}$ typically also includes the (co)variance matrix used for generating the noise $\boldsymbol{\epsilon}_t$. In robot learning, the three main policy representations are linear policies, radial basis function networks, and dynamic movement primitives [71].

Linear Policies. Linear controllers are the most simple time-independent representation. The policy π is a linear policy

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{1.6}$$

where $\boldsymbol{\phi}$ is a basis function vector. This policy only depends linearly on the policy parameters. However, specifying the basis functions by hand is typically a difficult task, and, hence, the application of linear controllers is limited to problems where appropriate basis functions are known, e.g., for balancing tasks, the basis functions are typically given by the state variables of the robot.

Radial Basis Functions Networks. A typical nonlinear time-independent policy representation is a radial basis function (RBF) network. An RBF policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{x})$ is given as

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}), \quad \boldsymbol{\phi}_i(\boldsymbol{x}) = \exp\left(-\tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \boldsymbol{D}_i (\boldsymbol{x} - \boldsymbol{\mu}_i)\right), \tag{1.7}$$

where $\boldsymbol{D}_i = \mathrm{diag}(\boldsymbol{d}_i)$ is a diagonal matrix. Unlike in the linear policy case, the parameters $\boldsymbol{\beta} = \{\boldsymbol{\mu}_i, \boldsymbol{d}_i\}_{i=1,\ldots,n}$ of the basis functions themselves are now considered as free parameters that need to be learned.

Hence, the parameter vector $\boldsymbol{\theta}$ of the policy is given by $\boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{\beta}\}$. While RBF-networks are powerful policy representations, they are also difficult to learn due to the high number of nonlinear parameters. Furthermore, as RBF networks are local representations, they are hard to scale to high-dimensional state spaces.

Dynamic Movement Primitives. Dynamic Movement Primitives (DMPs) are the most widely used time-dependent policy representation in robotics [71, 31]. DMPs use non-linear dynamical systems for generating the movement of the robot. The key principle of DMPs is to a use a linear spring-damper system which is modulated by a non-linear forcing function f_t, i.e.,

$$\ddot{y}_t = \tau^2 \alpha_y (\beta_y (g - y_t) - \dot{y}_t) + \tau^2 f_t \,, \tag{1.8}$$

where the variable y_t directly specifies the desired joint position of the robot. The parameter τ is the time-scaling coefficient of the DMP, the coefficients α_y and β_y define the spring and damping constants of the spring-damper system and the goal-parameter g is the unique point-attractor of the spring-damper system. Note that the spring-damper system is equivalent to a standard linear PD-controller that operates on a linear system with zero desired velocity, i.e.,

$$\ddot{y}_t = k_p (g - y_t) - k_d \dot{y}_t \,,$$

where the P-gain is given by $k_p = \tau^2 \alpha_y \beta_y$ and the D-gain by $k_d = \tau^2 \alpha_y$. The forcing function f_t changes the goal attractor g of the linear PD-controller.

One key innovation of the DMP approach is the use of a phase variable z_t to scale the execution speed of the movement. The phase variable evolves according to $\dot{z} = -\tau \alpha_z z$. It is initially set to $z = 1$ and exponentially converges to 0 as $t \to \infty$. The parameter α_z specifies the speed of the exponential decline of the phase variable. The variable τ can be used to temporally scale the evolution of the phase z_t, and, thus, the evolution of the spring-damper system as shown in Equation (1.8). For each degree of freedom, an individual spring damper system, and, hence, and individual forcing function f_t is used. The function f_t depends on the phase variable, i.e., $f_t = f(z_t)$ and is constructed by the weighted sum of K basis functions ϕ_i

$$f(z) = \frac{\sum_{i=1}^{K} \phi_i(z) w_i}{\sum_{i=1}^{K} \phi_i(z)} z, \qquad \phi_i(z) = \exp\left(-\tfrac{1}{2\sigma_i^2}(z - c_i)^2\right). \quad (1.9)$$

The parameters w_i are denoted as 'shape-parameters' of the DMP as they modulate the acceleration profile, and, hence, indirectly specify the shape of the movement. From Equation (1.9), we can see that the basis-functions are multiplied with the phase variable z, and, hence, f_t vanishes as $t \to \infty$. Consequently, the non-linear dynamical system is globally stable as it behaves like a linear spring damper system for $t \to \infty$. From this argument, we can also conclude that the goal parameter g specifies the final position of the movement while the shape parameters w_i specify how to reach this final position.

Integrating the dynamical systems for each DoF results in a desired trajectory $\boldsymbol{\tau}^* = \{\boldsymbol{y}_t\}_{t=0\ldots T}$ that is, subsequently, followed by feedback control laws [56]. The policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ that is specified by a DMP, directly controls the acceleration of the joint, and, hence, is given by

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) = \tau^2 \alpha_y(\beta_y(g - y_t) - \dot{y}_t) + \tau^2 f(z_t).$$

Note that the DMP policy is linear in the shape parameters $\boldsymbol{w}$ and the goal attractor g, but non-linear in the time scaling constant τ.

The parameters $\boldsymbol{\theta}$ used for learning a DMP are typically given by the weight parameters w_i, but might also contain the goal parameters g as well as the temporal scaling parameter τ. In addition, the DMP approach has been extended in [36] such that the desired final velocity $\dot{g}$ of the joints can also be modulated. Such modulation is, for example, useful for learning hitting movements in robot table-tennis. Typically, $K = 5$ to 20 basis functions are used, i.e., 5 to 20 shape weights per degree of freedom of the robot are used.

Miscellaneous Representations. Other representations that have been used in the literature include central pattern generators for robot walking [24] and feed-forward neural networks, which have been used mainly in simulation [30, 89].

1.4 Outline

The structure of this monograph is as follows: In Section 2, we give a detailed overview of model-free policy search methods, where we classify policy search algorithms according to their policy evaluation, policy update, and exploration strategy. For the policy update strategies, we will follow the taxonomy in Figure 1.1 and discuss policy gradient methods, EM-based approaches, information-theoretic approaches. Additionally, we will discuss miscellaneous important methods such as stochastic optimization and policy search approaches based on the path integral theory. Policy search algorithms can either use a step-based or episode-based policy evaluation strategy. Most policy update strategies presented in Figure 1.1 can be used for both, step-based and episode-based policy evaluation. We will present both types of algorithms if they have been introduced in the literature. Subsequently, we will discuss different exploration strategies for model-free policy search and conclude this section with robot applications of model-free policy search. Section 3 surveys model-based policy search methods in robotics. Here, we introduce two models that are commonly used in policy search: locally weighted regression and Gaussian processes. Furthermore, we detail stochastic and deterministic inference algorithms to compute a probability distribution $p_\pi(\boldsymbol{\tau})$ over trajectories (see the red boxes in Figure 1.1). We conclude this section with examples of model-based policy search methods and their application to robotic systems. In Section 4, we give recommendations for the practitioner and conclude this monograph.

2

Model-free Policy Search

Model-free policy search (PS) methods update the policy directly based on sampled trajectories $\boldsymbol{\tau}^{[i]}$, where i denotes the index of the trajectory, and the obtained immediate rewards $r_0^{[i]}, r_1^{[i]}, \ldots, r_T^{[i]}$ for the trajectories. Model-free PS methods try to update the parameters $\boldsymbol{\theta}$ such that trajectories with higher rewards become more likely when following the new policy, and, hence, the average return

$$J_{\boldsymbol{\theta}} = \mathbb{E}[R(\boldsymbol{\tau})|\boldsymbol{\theta}] = \int R(\boldsymbol{\tau}) p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) d\boldsymbol{\tau} \tag{2.1}$$

increases. Learning a policy is often easier than learning a model of the robot and its environment, and, hence, model-free policy search methods are used more frequently than model-based policy search methods. We categorize model-free policy search approaches based on their policy evaluation strategies, their policy update strategies [58, 57] and their exploration strategies [67, 38].

The exploration strategy determines how new trajectories are created for the subsequent policy evaluation step. The exploration strategy is essential for efficient model-free policy search, as, we need variability in the generated trajectories to determine the policy update, but an excessive exploration is also likely to damage the robot. Most model-free

Algorithm 1 Model-Free Policy Search

```
repeat
    Explore: Generate trajectories $\boldsymbol{\tau}^{[i]}$ using policy $\pi_k$
    Evaluate: Assess quality of trajectories or actions
    Update: Compute $\pi_{k+1}$ given trajectories $\boldsymbol{\tau}^{[i]}$ and evaluations
until Policy converges $\pi_{k+1} \approx \pi_k$
```

methods therefore use a stochastic policy for exploration which explores only locally. Exploration strategies can be categorized into step-based and episode-based exploration strategies. While step-based exploration uses an exploratory action in each time step, episode-based exploration directly changes the parameter vector $\boldsymbol{\theta}$ of the policy only at the start of the episode.

The policy evaluation strategy decides how to evaluate the quality of the executed trajectories. Here we can again distinguish between step-based and episode-based evaluations. Step-based evaluation strategies decompose the trajectory $\boldsymbol{\tau}$ in its single steps $(\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \dots)$ and aim at evaluating the quality of single actions. In comparison, episode-based evaluation directly uses the returns of the whole trajectories to evaluate the quality of the used policy parameters $\boldsymbol{\theta}$.

Finally, the policy update strategy uses the quality assessment of the evaluation strategy to determine the policy update. Update strategies can be classified according to the optimization method employed by the PS algorithm. While the most common update strategies are based on gradient ascent, resulting in policy gradient methods [89, 58, 61], inference-based approaches use expectation maximization [38, 48] and information theoretic approaches [57, 17] use insights from information theory to update the policy. We will also cover additional important methods such as path-integral approaches and stochastic optimization. Model-free policy search can be applied to policies with a moderate number of parameters, i.e., up to a few hundred parameters. Most applications use linear policy representations such as linear controllers or dynamical movement primitives that have been discussed in Section 1.3.

In the following section, we will discuss the used exploration strategies in current algorithms. Subsequently, we will cover policy evaluation strategies in more detail. Finally, we will review policy update methods such as policy gradients, inference/EM-based, and information theoretic policy updates as well as update strategies based on path integrals. Many policy update strategies have been implemented for both policy evaluation approaches, and, hence, we will discuss the combinations that have been explored so far. We will conclude with presenting the most interesting experimental results for policy learning for robots.

2.1 Exploration Strategies

The exploration strategy is used to generate new trajectory samples $\boldsymbol{\tau}^{[i]}$ which are subsequently evaluated by the policy evaluation strategy and used for the policy update. An efficient exploration is, therefore, crucial for the performance of the resulting policy search algorithm. All exploration strategies considered for model-free policy search are local and use stochastic policies to implement exploration. Typically, Gaussian policies are used to model such stochastic policies.

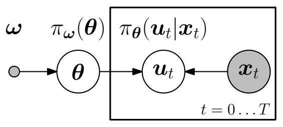
We distinguish between exploration in action space versus exploration in parameter space, step-based versus episode-based exploration strategies and correlated versus uncorrelated exploration noise.

2.1.1 Exploration in Action Space versus Exploration in Parameter Space

Exploration in the action space is implemented by adding an exploration noise $\boldsymbol{\epsilon}_{\boldsymbol{u}}$ directly to the executed actions, i.e. $\boldsymbol{u}_t = \boldsymbol{\mu}(\boldsymbol{x}, t) + \boldsymbol{\epsilon}_{\boldsymbol{u}}$. The exploration noise is in most cases sampled independently for each time step from a zero-mean Gaussian distribution with covariance $\boldsymbol{\Sigma}_{\boldsymbol{u}}$. The policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ is, therefore, given as

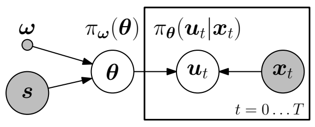
$$\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu}_{\boldsymbol{u}}(\boldsymbol{x}, t), \boldsymbol{\Sigma}_{\boldsymbol{u}}).$$

Exploration in the action space is one of the first exploration strategies used in the literature [89, 79, 9, 62] and used for most policy gradient approaches such as the REINFORCE algorithm [89] or the eNAC algorithm [61].



$\boldsymbol{\omega}$
$\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$
$\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)$
$\boldsymbol{\theta}$
$\boldsymbol{u}_t$
$\boldsymbol{x}_t$
$t = 0 \ldots T$

(a) Learning an upper-level policy.



$\boldsymbol{\omega}$
$\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$
$\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)$
$\boldsymbol{s}$
$\boldsymbol{\theta}$
$\boldsymbol{u}_t$
$\boldsymbol{x}_t$
$t = 0 \ldots T$

(b) Learning an upper-level policy for multiple contexts.

Fig. 2.1 (a) Graphical model for learning an upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$. The upper level policy chooses the parameters $\boldsymbol{\theta}$ of the lower-level policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ that is executed on the robot. The parameters of the upper-level policy are given by $\boldsymbol{\omega}$. (b) Learning an upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ for multiple contexts $\boldsymbol{s}$. The context is used to select the parameters $\boldsymbol{\theta}$, but typically not be the lower-level policy itself. The lower-level policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ is typically modeled as a deterministic policy in both settings.

Exploration strategies in parameter space perturb the parameter vector $\boldsymbol{\theta}$. This exploration can either only be added in the beginning of an episode, or, a different perturbation of the parameter vector can be used at each time step [67, 38].

Learning Upper-Level Policies Both approaches can be formalized with the concept of an upper level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ which selects the parameters of the actual control policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ of the robot. Hence, we will denote the latter in this hierarchical setting as lower-level policy. The upper level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ is typically modeled as a Gaussian distribution $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$. The lower level control policy $\boldsymbol{u} = \pi_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$ is typically modeled as deterministic policy as exploration only takes place in the parameter space of the policy.

Instead of directly finding the parameters $\boldsymbol{\theta}$ of the lower-level policy, we want to find the parameter vector $\boldsymbol{\omega}$ which now defines a distribution over $\boldsymbol{\theta}$. Using a distribution over $\boldsymbol{\theta}$ has the benefit that we can use this distribution to directly explore in parameter space. The optimization problem for learning upper-level policies can be formalized as maximizing

$$J_{\boldsymbol{\omega}} = \int_{\boldsymbol{\theta}} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}|\boldsymbol{\theta}) R(\boldsymbol{\tau}) d\boldsymbol{\tau} d\boldsymbol{\theta} = \int_{\boldsymbol{\theta}} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}) R(\boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{2.2}$$

The graphical model for learning an upper-level policy is given in Figure 2.1(a).

In the case of using a different parameter vector for each time step, typically, a linear control policy $u_t = \boldsymbol{\phi}_t(\boldsymbol{x})^T\boldsymbol{\theta}$ is used. We can also rewrite the deterministic lower level control policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$ in combination with the upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ as a single, stochastic policy

$$\pi_{\boldsymbol{\theta}}(u_t|\boldsymbol{x}_t, t) = \mathcal{N}\left(u_t|\boldsymbol{\phi}_t(\boldsymbol{x})^T\boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\phi}_t(\boldsymbol{x})^T\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{\phi}_t(\boldsymbol{x})\right), \tag{2.3}$$

which follows from the properties of the expectation and the variance operators. Such exploration strategy is, for example, applied by the PoWER [38] and PI^2 [81] algorithms and was also suggested to be used for policy gradient algorithms in [67].

In contrast to exploration in action space, exploration in parameter space is able to use more structured noise and can adapt the variance of the exploration noise in dependence of the state features $\boldsymbol{\phi}_t(\boldsymbol{x})$.

2.1.2 Episode-based versus Step-based Exploration

Step-based exploration strategies use different exploration noise at each time-step and can explore either in action space or in parameter space as we know from the discussion in the previous section. Episode-based exploration strategies use exploration noise only at the beginning of the episode, which naturally leads to an exploration in parameter space. Typically, episode-based exploration strategies are used in combination with episode-based policy evaluation strategies which are covered in the next section. However, episode-based exploration strategies are also realizable with step-based evaluation strategies such as with the PoWER [38] or with the PI^2 [81] algorithm.

Step-based exploration strategies can be problematic as they might produce action sequences which are not reproducible by the noise-free control law, and, hence, might again affect the quality of the policy updates. Furthermore, the effects of the perturbations are difficult to estimate as they are typically washed out by the system dynamics which acts as a low pass filter. Moreover, a step-based exploration strategy causes a large parameter variance which grows with the number of time steps. Such exploration strategies may even damage the robot as random exploration in every time step leads to jumps in the controls of the robot. Hence, fixing exploration for the whole episode or only slowly vary the exploration by low-pass filtering the noise, is often beneficial in real robot applications. On the other hand, the stochasticity of the control policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ also smoothens out the expected return, and, hence, in our experience, policy search is sometimes less prone to getting stuck in local minima using step-based exploration.

Episode-based exploration always produces action sequences which can be reproduced by the noise free control law. Fixing the exploration noise in the beginning of the episode might also decrease the variance of the quality assessment estimated by the policy evaluation strategy, and, hence, might produce more reliable policy updates [77].

2.1.3 Uncorrelated versus Correlated Exploration

As most policies are represented as Gaussian distributions, uncorrelated exploration noise is obtained by using a diagonal covariance matrix while we can achieve correlated exploration by maintaining a full representation of the covariance matrix. Exploration strategies in action space typically use a diagonal covariance matrix. For exploration strategies in parameter space, many approaches can also be used to update the full covariance matrix of the Gaussian policy. Such an approach was first introduced by the stochastic optimization algorithm CMA-ES [28] and was later also incorporated into more recent policy search approaches such as REPS [57, 17], PoWER [38], and PI^2 [77].

Using the full covariance matrix often results in a considerably increased learning speed for the resulting policy search algorithm [77]. Intuitively, the covariance matrix can be interpreted as a second order term. Similar to the Hessian in second order optimization approaches, it regulates the step-width of the policy update for each direction of the parameter space. However, estimating the full covariance matrix can also be difficult [68] if the parameter space becomes high dimensional ($|\boldsymbol{\Theta}| > 50$) as the covariance matrix has $O(|\boldsymbol{\Theta}|^2)$ elements. In this case, too many samples are needed for an accurate estimate of the covariance matrix.

2.1.4 Updating the Exploration Distribution

Many model-free policy search approaches also update the exploration distribution, and, hence, the covariance of the Gaussian policy. Updating the exploration distribution often improves the performance as different exploration rates can be used throughout the learning process. Typically, a large exploration rate can be used in the beginning of learning which is then gradually decreased to fine tune the policy parameters. In general, the exploration rate tends to be decreased too quickly for many algorithms, and, hence, the exploration policy might collapse to almost a single point. In this case, the policy update might stop improving prematurely. This problem can be alleviated by the use of an information theoretic update metric, which limits the relative entropy between the new and the old trajectory distribution. Such an information theoretic measure is, for example, used by the natural policy gradient methods as well as by the REPS algorithm [57, 17]. Peters and Schaal [61] showed that, due to the bounded relative entropy to the old trajectory distribution, the exploration does not collapse prematurely, and hence, a more stable learning progress can be achieved. Still, the exploration policy might collapse prematurely if initialized only locally. Some approaches artificially add an additional variance term to the covariance matrix of the exploration policy after each policy update to sustain exploration [77], however, a principled way of adapting the exploration policy in such situations is missing so far in the literature.

2.2 Policy Evaluation Strategies

Policy evaluation strategies are used to assess the quality of the executed policy. Policy search algorithms either try to assess the quality of single state-action pairs $\boldsymbol{x}_t, \boldsymbol{u}_t$, which we will refer to as step-based evaluations, or the quality of a parameter vector $\boldsymbol{\theta}$ that has been used during the whole episode, which we will refer to as episode-based policy evaluation. The policy evaluation strategy is used to transform sampled trajectories $\boldsymbol{\tau}^{[i]}$ into a data-set $\mathcal{D}$ that contains samples of either the state-action pairs $\boldsymbol{x}_t^{[i]}, \boldsymbol{u}_t^{[i]}$ or the parameter vectors $\boldsymbol{\theta}^{[i]}$ and an evaluation of these samples, as will be described in this section. The data-set $\mathcal{D}$ is subsequently processed by the policy update strategy to determine the new policy.

2.2.1 Step-Based Policy Evaluation

In step-based policy evaluation, we decompose the sampled trajectories $\boldsymbol{\tau}^{[i]}$ into its single time steps $\boldsymbol{x}_t^{[i]}, \boldsymbol{u}_t^{[i]}$, and estimate the quality of the single actions. The quality of an action is given by the expected accumulated future reward when executing $\boldsymbol{u}_t^{[i]}$ in state $\boldsymbol{x}_t^{[i]}$ at time step t and subsequently following policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$,

$$Q_t^{[i]} = Q_t^{\pi}\left(\boldsymbol{x}_t^{[i]}, \boldsymbol{u}_t^{[i]}\right) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{h=t}^{T} r_h(\boldsymbol{x}_h, \boldsymbol{u}_h)\middle| \boldsymbol{x}_t = \boldsymbol{x}_t^{[i]}, \boldsymbol{u}_t = \boldsymbol{u}_t^{[i]}\right],$$

which corresponds to the state-action value function Q^{π}. However, estimating the state-action value function is a difficult problem in high-dimensional continuous spaces and often suffers from approximation errors or a bias induced by the bootstrapping approach used by most value function approximation methods. Therefore, most policy search methods rely on Monte-Carlo estimates of $Q_t^{[i]}$ instead of using value function approximations. Monte-Carlo estimates are unbiased, however, they typically exhibit a high variance. Fortunately, most methods can cope with noisy estimates of $Q_t^{[i]}$, and, hence, solely the reward to come for the current trajectory $\boldsymbol{\tau}^{[i]}$ is used $Q_t^{[i]} \approx \sum_{h=t}^{T} r_h^{[i]}$, which avoids the additional averaging that would be needed for an accurate Monte-Carlo estimate. Algorithms based on step-based policy evaluation use a data set $\mathcal{D}_{\text{step}} = \{\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}, Q^{[i]}\}$ to determine the policy update step. Some step-based policy search algorithms [89, 58] also use the returns $R^{[i]} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{h=0}^{T} r_h^{[i]}\right]$ of the whole episode as evaluation for the single actions of the episode. However, as the estimate of the returns suffers from a higher variance than the reward to come, such a strategy is not preferable. Pseudo-code for a general step-based policy evaluation algorithm is given in Algorithm 2.

Algorithm 2 Policy Search with Step-Based Policy Evaluation

```
repeat
    Exploration:
        Create samples $\boldsymbol{\tau}^{[i]} \sim \pi_{\boldsymbol{\theta}_k}(\boldsymbol{\tau})$ following $\pi_{\boldsymbol{\theta}_k}(\boldsymbol{u}|\boldsymbol{x})$, $i = 1 \ldots N$
    Policy Evaluation:
        Evaluate actions: $Q_t^{[i]} = \sum_{h=t}^{T} r_h^{[i]}$ for all $t$ and all $i$
        Compose data set: $\mathcal{D}_{\text{step}} = \left\{\boldsymbol{x}_t^{[i]}, \boldsymbol{u}_t^{[i]}, Q_t^{[i]}\right\}_{i=1\ldots N,\, t=1\ldots T-1}$
    Policy Update:
        Compute new policy parameters $\boldsymbol{\theta}_{k+1}$ using $\mathcal{D}$.
        Algorithms: REINFORCE, G(PO)MDP, NAC, eNAC,
            PoWER, PI$^2$
until Policy converges $\boldsymbol{\theta}_{k+1} \approx \boldsymbol{\theta}_k$
```

2.2.2 Episode-Based Policy Evaluation

Episode-based update strategies [77, 78, 17] directly use the expected return $R^{[i]} = R\left(\boldsymbol{\theta}^{[i]}\right)$ to evaluate the quality of a parameter vector $\boldsymbol{\theta}^{[i]}$. Typically , the expected return is given by the sum of the future immediate rewards, i.e.,

$$R\left(\boldsymbol{\theta}^{[i]}\right) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{t=0}^{T} r_t | \boldsymbol{\theta} = \boldsymbol{\theta}^{[i]}\right]. \tag{2.4}$$

However, episode-based algorithms are not restricted to this structure of the return, but can use any reward function $R\left(\boldsymbol{\theta}^{[i]}\right)$ which depends on the resulting trajectory of the robot. For example, when we want to learn to throw a ball to a desired target location, the reward $R\left(\boldsymbol{\theta}^{[i]}\right)$ can intuitively be defined as the negative minimum distance of the ball to the target location [41]. Such reward function can not be described by a sum of immediate rewards.

The expected return $R^{[i]}$ for $\boldsymbol{\theta}^{[i]}$ can be estimated by performing multiple roll-outs on the real system. However, in order to avoid such an expensive operation, a few approaches [68, 38] can cope with noisy estimates of $R^{[i]}$, and, hence, can directly use the return $\sum_{t=0}^{T} r_t^{[i]}$ of a single trajectory $\boldsymbol{\tau}^{[i]}$ to estimate $R^{[i]}$. Other algorithms, such as stochas-

Algorithm 3 Episode-based Policy Evaluation for Learning an Upper-Level Policy

```
repeat
    Exploration:
        Sample parameter vector $\boldsymbol{\theta}^{[i]} \sim \pi_{\boldsymbol{\omega}_k}(\boldsymbol{\theta})$, $i = 1 \dots N$
        Sample trajectory $\boldsymbol{\tau}^{[i]} \sim p_{\boldsymbol{\theta}^{[i]}}(\boldsymbol{\tau})$
    Policy Evaluation:
        Evaluate policy parameters $R^{[i]} = \sum_{t=0}^{T} r_t^{[i]}$
        Compose data set $\mathcal{D}_{\text{ep}} = \left\{\boldsymbol{\theta}^{[i]}, R^{[i]}\right\}_{i=1\dots N}$
    Policy Update:
        Compute new policy parameters $\boldsymbol{\omega}_{k+1}$ using $\mathcal{D}_{\text{ep}}$
        Algorithms: Episode-based REPS, Episode-based PI$^2$
            PEPG, NES, CMA-ES, RWR
until Policy converges $\boldsymbol{\omega}_{k+1} \approx \boldsymbol{\omega}_k$
```

tic optimizers, require a more accurate estimate of $R^{[i]}$, and, thus, either require multiple roll-outs, or suffer from a bias in the subsequent policy update step. Episode-based policy evaluation produces a data-set $\mathcal{D}_{\text{ep}} = \{\boldsymbol{\theta}^{[i]}, R^{[i]}\}_{i=1\dots N}$, which is subsequently used for the policy updates. Episode-based policy evaluation is typically connected with parameter-based exploration strategies, and, hence, such algorithms can be formalized by the problem of learning an upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$, see Section 2.1.1. The general algorithm for policy search with episode-based policy evaluation is given in Algorithm 3.

An underlying problem of episode-based evaluation is the variance of the $R^{[i]}$ estimates. The variance depends on the stochasticity of the system, the stochasticity of the policy, and the number of time steps, consequently, for a high number of time-steps and and highly stochastic systems, step-based algorithms should be preferred. In order to reduce the variance, the policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ is often modeled as a deterministic policy and exploration is directly performed in parameter space.

2.2.3 Comparison of Step- and Episode-based Evaluation

Step-based policy evaluation exploits the structure that the return is typically composed of the sum of the immediate rewards. Single actions can now be evaluated by the reward to come in that episode, instead of the whole reward of the episode, and, hence, the variance of the evaluation can be significantly reduced as the reward to come contains less random variables as the total reward of the episode. In addition, as we evaluate single actions instead of the whole parameter vector, the evaluated samples can be used more efficiently as several parameter-vectors $\boldsymbol{\theta}$ might produce a similar action $\boldsymbol{u}_t$ at time step t. Most policy search algorithms, such as the common policy gradient algorithms [89, 61], the PoWER [38] algorithm or the PI^2 [81] algorithm, employ such a strategy. A drawback of most step-based updates is that they often rely on a linear parametrization of the policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$. They also cannot be applied if the reward is not decomposable into isolated time steps.

Episode-based policy evaluation strategies do not decompose the returns, and, hence, might suffer from a large variance of the estimated returns. However, episode-based policy evaluation strategies typically employ more sophisticated exploration strategies which directly explore in the parameter space of the policy [17, 77, 29], and, thus, can often compete with their step-based counter-parts. So far, there is no clear answer as to which of the strategies should be preferred. The choice of the methods often depends on the problem at hand.

2.3 Important Extensions

In this section we will cover two important extensions of model-free policy search, generalization to multiple tasks and learning multiple solutions to the same task. We will introduce the relevant concepts for both extensions, however, the detailed algorithms will be covered in Section 2.4 which covers the policy update strategies.

2.3.1 Generalization to Multiple Tasks

For generalizing the learned policies to multiple tasks, so far, mainly episode-based policy evaluation strategies have been used which learn an upper level policy. We will characterize a task by a context vector $\boldsymbol{s}$. The context describes all variables which do not change during the execution of the task but might change from task to task. For example, the context $\boldsymbol{s}$ can describe the objectives of the agent or physical properties such as a mass to lift. The upper level policy is extended to generalize the lower-level policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ to different tasks by conditioning the upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ on the context $\boldsymbol{s}$. The problem of learning $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ can be characterized by maximizing the expected returns over all contexts, i.e.,

$$J_{\boldsymbol{\omega}} = \int_{\boldsymbol{s}} \mu(\boldsymbol{s}) \int_{\boldsymbol{\theta}} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}|\boldsymbol{\theta}, \boldsymbol{s}) R(\boldsymbol{\tau}, \boldsymbol{s}) d\boldsymbol{\tau} d\boldsymbol{\theta} d\boldsymbol{s} \tag{2.5}$$

$$= \int_{\boldsymbol{s}} \mu(\boldsymbol{s}) \int_{\boldsymbol{\theta}} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) R(\boldsymbol{\theta}, \boldsymbol{s}) d\boldsymbol{\theta} d\boldsymbol{s}, \tag{2.6}$$

where $R(\boldsymbol{\theta}, \boldsymbol{s}) = \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}|\boldsymbol{\theta}, \boldsymbol{s}) R(\boldsymbol{\tau}, \boldsymbol{s}) d\boldsymbol{\tau}$ is the expected return for executing the lower-level policy with parameter vector $\boldsymbol{\theta}$ in context $\boldsymbol{s}$ and $\mu(\boldsymbol{s})$ is the distribution over the contexts. The trajectory distribution $p(\boldsymbol{\tau}|\boldsymbol{\theta}, \boldsymbol{s})$ can now also depend on the context, as the context can contain physical properties of the environment. We also extend the data set $\mathcal{D}_{\text{ep}} = \left\{\boldsymbol{s}^{[i]}, \boldsymbol{\theta}^{[i]}, R^{[i]}\right\}_{i=1\ldots N}$ used for updating the policy, which now also includes the corresponding context $\boldsymbol{s}^{[i]}$ that have been active for executing the lower-level policy with parameters $\boldsymbol{\theta}^{[i]}$. The graphical model for learning an upper-level policies with multiple contexts is given in Figure 2.1(b) and the general algorithm is given in Algorithm 4. Algorithms that can generalize the lower level policy to multiple contexts include the Reward Weighted Regression (RWR) algorithm [59] the Cost-Regularized Regression (CrKR) algorithm [37] and the episode-based relative entropy policy search (REPS) algorithm [17]. RWR and CrKR are covered in Section 2.4.2.3 and REPS in Section 2.4.3.1.

2.3.2 Learning Multiple Solutions for a Single Motor Task

Many motor tasks can be solved in multiple ways. For example, for returning a tennis ball, in many situations we can either use a backhand or fore-hand stroke. Hence, it is desirable to find algorithms

Algorithm 4 Learning an Upper-Level Policy for multiple Tasks

```
repeat
    Exploration:
        Sample context $\boldsymbol{s}^{[i]} \sim \mu(\boldsymbol{s})$
        Sample parameter vector $\boldsymbol{\theta}^{[i]} \sim \pi_{\boldsymbol{\omega}_k}(\boldsymbol{\theta}|\boldsymbol{s}^{[i]})$, $i = 1 \dots N$
        Sample trajectory $\boldsymbol{\tau}^{[i]} \sim p_{\boldsymbol{\theta}^{[i]}}(\boldsymbol{\tau}|\boldsymbol{s}^{[i]})$
    Policy Evaluation:
        Evaluate policy parameters $R^{[i]} = \sum_{t=0}^{T} r_t^{[i]}$
        Compose data set $\mathcal{D}_{\text{ep}} = \left\{\boldsymbol{\theta}^{[i]}, \boldsymbol{s}^{[i]}, R^{[i]}\right\}_{i=1\dots N}$
    Policy Update:
        Compute new policy parameters $\boldsymbol{\omega}_{k+1}$ using $\mathcal{D}_{\text{ep}}$
        Algorithms: Episode-based REPS, CRKR, PEPG, RWR
until Policy converges $\boldsymbol{\omega}_{k+1} \approx \boldsymbol{\omega}_k$
```

that can learn and represent multiple solutions for one task. Such approaches increase the robustness of the learned policy in the case of slightly changing conditions, as we can resort to backup-solutions. Moreover, many policy search approaches have problems with multi-modal solution spaces. For example, EM-based or information-theoretic approaches use a weighted average of the samples to determine the new policy. Therefore, these approaches average over several modes, which can considerably decrease the quality of the resulting policy. Such problems can be resolved by using policy updates which are not based on weighted averaging, see Section 2.4.2.4, or by using a mixture model to directly represent several modes in the parameter space [17, 66]. We will discuss such an approach, which is based on episode based REPS in Section 2.4.3.3.

2.4 Policy Update Strategies

In this section, we will describe different policy update strategies used in policy search, such as policy gradient methods, expectation-maximization based methods, information theoretic methods, and policy updates which can by derived from the path-integral theory. In the case where the policy update method has been introduced for both step-based and episode-based policy evaluation strategies, we will present both resulting algorithms. Policy updates for the step-based evaluation strategy use the data set $\mathcal{D}_{\text{step}}$ to determine the policy update while algorithms based on episode-based policy updates employ the data set $\mathcal{D}_{\text{ep}}$. We will qualitatively compare the algorithms with respect to their sample efficiency, the number of algorithmic parameters that have to be tuned by the user, the type of reward-function that can be employed and also how safe it is to apply the method on a real robot. Methods which are safe to apply on a real robot should not allow big jumps in the policy updates, as such jumps might result in unpredictable behavior which might damage the robot. We will now discuss the different policy update strategies.

Whenever it is possible, we will describe the policy search methods for the episodic reinforcement learning formulation with time-dependent policies as most robot learning task are episodic and not infinite horizon tasks. However, most of the derivations also hold for the infinite horizon formulation if we introduce a discount factor γ for the return and cut the trajectories after a horizon of T time steps, where T needs to be sufficiently large such that the influence of future time-steps with $t > T$ vanishes as γ^T approaches zero.

2.4.1 Policy Gradient Methods

Policy gradient (PG) methods use gradient-ascent for maximizing the expected return $J_{\boldsymbol{\theta}}$. In gradient ascent, the parameter update direction is given by the gradient $\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}$ as it points in the direction of steepest ascent of the expected return. The policy gradient update is therefore given by

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}},$$

where α is a user-specified learning rate and the policy gradient is given by

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} = \int_{\boldsymbol{\tau}} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}. \tag{2.7}$$

We will now discuss different ways to estimate the gradient $\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}$.

2.4.1.1 Finite Difference Methods

The finite difference policy gradient [39, 58] is the simplest way of obtaining the policy gradient. It is typically used with the episode-based evaluation strategy. The finite difference method estimates the gradient by applying small perturbations $\delta\boldsymbol{\theta}^{[i]}$ to the parameter vector $\boldsymbol{\theta}_k$. We can either perturb each parameter value separately or use a probability distribution with small variance to create the perturbations. For each perturbation, we obtain the change of the return $\delta R^{[i]} = R(\boldsymbol{\theta}_\mathrm{k} + \delta\boldsymbol{\theta}^{[i]}) - R(\boldsymbol{\theta}_\mathrm{k})$. For finite difference methods, the perturbations $\delta\boldsymbol{\theta}^{[i]}$ implement the exploration strategy in parameter space. However, the generation of the perturbations is typically not adapted during learning but predetermined by the user. The gradient $\nabla^{\mathrm{FD}}_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}$ can be obtained by using a first order Taylor-expansion of $J_{\boldsymbol{\theta}}$ and solving for the gradient in a least-squares sense, i.e., it is determined numerically from the samples as

$$\nabla^{\mathrm{FD}}_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} = (\delta\boldsymbol{\Theta}^T \delta\boldsymbol{\Theta})^{-1} \delta\boldsymbol{\Theta}^T \delta\boldsymbol{R}, \tag{2.8}$$

where $\delta\boldsymbol{\Theta} = \left[\delta\boldsymbol{\theta}^{[1]}, \dots, \delta\boldsymbol{\theta}^{[N]}\right]^T$ and $\delta\boldsymbol{R} = [\delta R^{[1]}, \dots, \delta R^{[N]}]$. Finite difference methods are powerful black-box optimizers as long as the evaluations $R(\boldsymbol{\theta})$ are not too noisy. From optimization, this method is also known as Least Square-Based Finite Difference (LSFD) scheme [75].

2.4.1.2 Likelihood-Ratio Policy Gradients

Likelihood-ratio methods were among the first policy search methods introduced in the early 1990s by Williams [89], and include the REINFORCE algorithm. These methods make use of the so called 'likelihood-ratio' trick that is given by the identity $\nabla p_{\boldsymbol{\theta}}(\boldsymbol{y}) = p_{\boldsymbol{\theta}}(\boldsymbol{y}) \nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{y})$[1]. Inserting the likelihood-ratio trick into the policy gradient from Equation (2.7) yields

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} = \int p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right], \tag{2.9}$$

[1] We can easily confirm this identity by using the chain rule to calculate the derivative of $\log p_{\boldsymbol{\theta}}(\boldsymbol{y})$, i.e., $\nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{y}) = \nabla p_{\boldsymbol{\theta}}(\boldsymbol{y}) / p_{\boldsymbol{\theta}}(\boldsymbol{y})$.

where the expectation over $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ is approximated by using a sum over the sampled trajectories $\boldsymbol{\tau}^{[i]} = (\boldsymbol{x}_0^{[i]}, \boldsymbol{u}_0^{[i]}, \boldsymbol{x}_1^{[i]}, \boldsymbol{u}_1^{[i]}, \dots)$.

Baselines. As the evaluation $R^{[i]}$ of a parameter $\boldsymbol{\theta}^{[i]}$ or the evaluation $Q_t^{[i]}$ of an action $\boldsymbol{u}^{[i]}$ is typically performed by inherently noisy Monte-Carlo estimates, the resulting gradient estimates are also afflicted by a large variance. The variance can be reduced by introducing a baseline b for the trajectory reward $R(\boldsymbol{\tau})$, i.e.,

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})(R(\boldsymbol{\tau}) - b)\right]. \tag{2.10}$$

Note that the policy gradient estimate remains unbiased as

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) b\right] = b \int_{\boldsymbol{\tau}} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) d\boldsymbol{\tau} = b \nabla_{\boldsymbol{\theta}} \int_{\boldsymbol{\tau}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) d\boldsymbol{\tau} = 0, \tag{2.11}$$

where we first applied the reverse of the 'likelihood-ratio' trick and subsequently the identity $\int_{\boldsymbol{\tau}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) d\boldsymbol{\tau} = 1$. Since the baseline b is a free parameter, we can choose it such that it minimizes the variance of the gradient estimate. We will denote the variance-minimizing baseline as the optimal baseline. As the likelihood gradient can be estimated in different ways, the corresponding optimal baseline will change with it. We now first discuss the step-based likelihood-ratio PG algorithms, and, discuss their optimal baselines if it is given in the literature. Subsequently, we will cover the episode-based likelihood-ratio variant.

Step-Based Likelihood-Ratio Methods

Step-based algorithms exploit the structure of the trajectory distribution, i.e.,

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = p(\boldsymbol{x}_1) \prod_{t=1}^{T} p(\boldsymbol{x}_{t+1} | \boldsymbol{x}_t, \boldsymbol{u}_t) \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t | \boldsymbol{x}_t, t)$$

to decompose $\nabla_{\boldsymbol{\theta}} \log p_{\theta}(\boldsymbol{\tau})$ into the single time steps. As the product is transformed into a sum by a logarithm, all terms which do not depend on the policy parameters $\boldsymbol{\theta}$ disappear during differentiation. Hence,

$\nabla_{\boldsymbol{\theta}} \log p_\theta(\boldsymbol{\tau})$ is given by

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t). \tag{2.12}$$

Equation (2.12) reveals a key result for policy gradients: $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ does not depend on the transition model $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$. Note that this result holds for any stochastic policy. However, for deterministic policies $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$, the gradient $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ includes

$$\nabla_{\boldsymbol{\theta}} p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)) = \frac{\partial p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)}{\partial \boldsymbol{u}_t} \frac{\partial \boldsymbol{u}_t}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{u}_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)},$$

and, hence, the transition model needs to be known. Consequently, stochastic policies play a crucial role for policy gradient methods.

The REINFORCE Algorithm. Equation (2.12) is used by one of the first PG algorithms introduced in the machine learning literature, the REINFORCE algorithm [89]. The REINFORCE policy gradient is given by

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{RF}} J_{\boldsymbol{\theta}} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)(R(\boldsymbol{\tau}) - b)\right], \tag{2.13}$$

where b denotes the baseline.

To minimize the variance of $\nabla_{\boldsymbol{\theta}}^{\mathrm{RF}} J_{\boldsymbol{\theta}}$, we estimate the optimal baseline b^{RF}. The optimal baseline also depends on which element h of the gradient $\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}$ we want to evaluate, and, hence needs to be computed for each dimension h separately. The optimal baseline b_h^{RF} for the REINFORCE algorithm minimizes the variance of $\nabla_{\boldsymbol{\theta}_h}^{\mathrm{RF}} J_{\boldsymbol{\theta}}$, i.e., it satisfies the condition

$$\begin{aligned}\frac{\partial}{\partial b}\mathrm{Var}\left[\nabla_{\boldsymbol{\theta}_h}^{\mathrm{RF}} J_{\boldsymbol{\theta}}\right] &= \frac{\partial}{\partial b}\left(\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[(\nabla_{\boldsymbol{\theta}_h}^{\mathrm{RF}} J_{\boldsymbol{\theta}})^2\right] - \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\nabla_{\boldsymbol{\theta}_h}^{\mathrm{RF}} J_{\boldsymbol{\theta}}\right]^2\right)\\ &= \frac{\partial}{\partial b}\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[(\nabla_{\boldsymbol{\theta}_h}^{\mathrm{RF}} J_{\boldsymbol{\theta}})^2\right] = 0, \end{aligned} \tag{2.14}$$

where the second term disappeared as the expected gradient is not affected by the baseline, see Equations (2.10) and (2.11). Solving this

Algorithm 5 REINFORCE

```
Input: policy parametrization $\boldsymbol{\theta}$,
        data-set $\mathcal{D} = \left\{ \boldsymbol{x}_{1:T}^{[i]}, \boldsymbol{u}_{1:T-1}^{[i]}, r_{1:T}^{[i]} \right\}_{i=1\ldots N}$
Compute returns: $R^{[i]} = \sum_{t=0}^{T} r_t^{[i]}$
for each dimension $h$ of $\boldsymbol{\theta}$ do
    Estimate optimal baseline:
        $$b_h^{\mathrm{RF}} = \frac{\sum_{i=1}^{N} \left( \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}} \left( \boldsymbol{u}_t^{[i]} \middle| \boldsymbol{x}_t^{[i]}, t \right) \right)^2 R^{[i]}}{\sum_{i=1}^{N} \left( \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}} \left( \boldsymbol{u}_t^{[i]} \middle| \boldsymbol{x}_t^{[i]}, t \right) \right)^2}$$
    Estimate derivative for dimension $h$ of $\boldsymbol{\theta}$:
        $$\nabla_{\boldsymbol{\theta}_h}^{\mathrm{RF}} J_{\boldsymbol{\theta}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}} \left( \boldsymbol{u}_t^{[i]} \middle| \boldsymbol{x}_t^{[i]}, t \right) (R^{[i]} - b_h^{\mathrm{RF}})$$
end for
Return $\nabla_{\boldsymbol{\theta}}^{\mathrm{RF}} J_{\boldsymbol{\theta}}$
```

equation for b yields

$$b_h^{\mathrm{RF}} = \frac{\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left[\left(\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}} \left(\boldsymbol{u}_t | \boldsymbol{x}_t, t \right) \right)^2 R(\boldsymbol{\tau}) \right]}{\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left[\left(\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}} \left(\boldsymbol{u}_t | \boldsymbol{x}_t, t \right) \right)^2 \right]}. \tag{2.15}$$

The REINFORCE algorithm with its optimal baseline is summarized in Algorithm 5.

The G(PO)MDP Algorithm. From Equation (2.13), we realize that REINFORCE uses the returns $R(\boldsymbol{\tau})$ of the whole episode as the evaluations of single actions despite using the step-based policy evaluation strategy. As already discussed before, the variance of the returns can grow with the trajectory length, and, hence, deteriorate the performance of the algorithm even if used with the optimal baseline. However, by decomposing the return in the rewards of the single time steps, we can use the observation that rewards from the past do not depend on actions in the future, and, hence, $\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\partial_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t) r_j\right] = 0$ for $j < t$[2]. If we look at a reward r_j of a single time-step j, we realize that we can neglect all derivatives of future actions. This intuition has been used for the G(PO)MDP algorithm [9, 10] to decrease the variance of policy gradient estimates. The policy gradient of G(PO)MDP is given by

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{GMDP}} J_{\boldsymbol{\theta}} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{j=0}^{T-1}\sum_{t=0}^{j} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)(r_j - b_j)\right], \qquad (2.16)$$

where b_j is a time-dependent baseline. The optimal baseline for the G(PO)MDP algorithm $b_{h,j}^{\mathrm{GMDP}}(\boldsymbol{x})$ for time step j and dimension h of $\boldsymbol{\theta}$ can be obtained similarly as for the REINFORCE algorithm and is given by

$$b_{h,j}^{\mathrm{GMDP}} = \frac{\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\left(\sum_{t=0}^{j} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)\right)^2 r_j\right]}{\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\left(\sum_{t=0}^{j} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)\right)^2\right]}. \qquad (2.17)$$

The G(PO)MDP algorithm is summarized in Algorithm 6.

The Policy Gradient Theorem Algorithm. Instead of using the returns $R(\boldsymbol{\tau})$ we can also use the expected reward to come at time step t, i.e., $Q_t^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t)$, to evaluate an action $\boldsymbol{u}_t$. Mathematically, such an evaluation can be justified by the same observation that has been used for the G(PO)MDP algorithm, i.e., that rewards are not correlated with future actions. Such evaluation is used by the Policy Gradient Theorem (PGT) algorithm [79], which states that

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}^{\mathrm{PG}} J_{\boldsymbol{\theta}} &= \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)\left(\sum_{j=t}^{T} r_j\right)\right] \\ &= \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t) Q_t^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t)\right]. \qquad (2.18)\end{aligned}$$

We can again subtract an arbitrary baseline $b_t(\boldsymbol{x})$ from $Q_t^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t)$, which now depends on the state $\boldsymbol{x}$ as well as on the time step.

[2] We can follow the same argument as in Equation (2.11) for introducing the baseline to prove this identity.

Algorithm 6 G(PO)MDP Algorithm

```
Input: Policy parametrization $\boldsymbol{\theta}$,
        Data set $\mathcal{D} = \left\{ \boldsymbol{x}_{1:T}^{[i]}, \boldsymbol{u}_{1:T-1}^{[i]}, r_{1:T-1}^{[i]} \right\}_{i=1\ldots N}$
for each time step $t = 0 \ldots T-1$ do
    for each dimension $h$ of $\boldsymbol{\theta}$ do
        Estimate optimal time-dependent baseline:
        $$b_{h,j}^{\text{GMDP}} = \frac{\sum_{i=1}^{N} \left( \sum_{t=0}^{j} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}} \left( \boldsymbol{u}_h^{[i]} \middle| \boldsymbol{x}_h^{[i]}, h \right) \right)^2 r_j^{[i]}}{\sum_{i=1}^{N} \sum_{t=0}^{T-1} \left( \sum_{t=0}^{j} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}} \left( \boldsymbol{u}_k^{[i]} \middle| \boldsymbol{x}_k^{[i]}, k \right) \right)^2}$$
    end for
    Estimate gradient for dimension $h$:
    $$\nabla_{\boldsymbol{\theta}_h}^{\text{GMDP}} J_{\boldsymbol{\theta}} = \sum_{i=1}^{N} \sum_{j=0}^{T-1} \left( \sum_{t=0}^{j} \nabla_{\boldsymbol{\theta}_h} \log \pi_{\boldsymbol{\theta}} \left( \boldsymbol{u}_t^{[i]} \middle| \boldsymbol{x}_t^{[i]}, t \right) \right) \left( r_j^{[i]} - b_{h,j}^{\text{GMDP}} \right)$$
end for
Return $\nabla_{\boldsymbol{\theta}}^{\text{GMDP}} J_{\boldsymbol{\theta}}$
```

While $Q_t^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ can be estimated by Monte-Carlo roll-outs, the PGT algorithm can be used in combination with function approximation as will be covered in Section 2.4.1.3.

Episode-Based Likelihood-Ratio Methods

Episode-based likelihood-ratio methods directly update the upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ for choosing the parameters $\boldsymbol{\theta}$ of the lower-level policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)$. They optimize the expected return $J_{\boldsymbol{\omega}}$, as defined in Equation (2.2). The likelihood gradient of $J_{\boldsymbol{\omega}}$ can be directly obtained by replacing $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ with $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ and $R(\boldsymbol{\tau})$ with $R(\boldsymbol{\theta}) = \int_{\boldsymbol{\tau}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$ in Equation (2.9), resulting in

$$\nabla_{\boldsymbol{\omega}}^{\text{PE}} J_{\boldsymbol{\omega}} = \mathbb{E}_{\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})} \left[\nabla_{\boldsymbol{\omega}} \log \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}) (R(\boldsymbol{\theta}) - b) \right]. \tag{2.19}$$

Such an approach was first introduced by [73, 74] with the Parameter Exploring Policy Gradient (PEPG) algorithm. The optimal base-line

Algorithm 7 Parameter Exploring Policy Gradient Algorithm

```
Input: Policy parametrization $\boldsymbol{\omega}$
    Data set $\mathcal{D} = \left\{\boldsymbol{\theta}^{[i]}, R^{[i]}\right\}_{i=1\ldots N}$
for each dimension $h$ of $\boldsymbol{\omega}$ do
    Estimate optimal baseline:
        $$b_h^{\text{PGPE}} = \frac{\sum_{i=1}^{N} \left(\nabla_{\boldsymbol{\omega}_h} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}^{[i]})\right)^2 R^{[i]}}{\sum_{i=1}^{N} \left(\nabla_{\boldsymbol{\omega}_h} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}^{[i]})\right)^2}$$
    Estimate derivative for dimension $h$ of $\boldsymbol{\omega}$:
        $$\nabla_{\boldsymbol{\omega}_h}^{\text{PE}} J_{\boldsymbol{\omega}} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\omega}} \pi_{\boldsymbol{\omega}_h}(\boldsymbol{\theta}^{[i]})(R^{[i]} - b_h^{\text{PGPE}})$$
end for
```

b_h^{PGPE} for the h-th element of the PEPG gradient is obtained similarly as for the REINFORCE algorithm and given by

$$b_h^{\text{PGPE}} = \frac{\mathbb{E}_{\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})}\left[\left(\nabla_{\boldsymbol{\omega}_h} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})\right)^2 R(\boldsymbol{\theta})\right]}{\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\left(\nabla_{\boldsymbol{\omega}_h} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})\right)^2\right]}. \tag{2.20}$$

The PEPG algorithm is summarized in Algorithm 7.

2.4.1.3 Natural Gradients

The natural gradient [3] is a well known concept from supervised learning for optimizing parametrized probability distributions $p_{\boldsymbol{\theta}}(\boldsymbol{y})$, where $\boldsymbol{y}$ is a random variable, which often achieves faster convergence than the traditional gradient. Traditional gradient methods typically use an Euclidean metric $\delta\boldsymbol{\theta}^T \delta\boldsymbol{\theta}$ to determine the direction of the update $\delta\boldsymbol{\theta}$, i.e., they assume that all parameter dimensions have similarly strong effects on the resulting distribution. However, already small changes in $\boldsymbol{\theta}$ might result in large changes of the resulting distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y})$. As the gradient estimation typically depends on $p_{\boldsymbol{\theta}}(\boldsymbol{y})$ due to the sampling process, the next gradient estimate can also change dramatically.

To achieve a stable behavior of the learning process, it is desirable to enforce that the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y})$ does not change too much in one update step. This intuition is the key concept behind the natural gradient which limits the distance between the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y})$ before and $p_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\boldsymbol{y})$ after the update. To measure the distance between $p_{\boldsymbol{\theta}}(\boldsymbol{y})$ and $p_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\boldsymbol{y})$, the natural gradient uses an approximation of the Kullback-Leibler (KL) divergence. The KL-divergence is a similarity measure of two distributions. It has been shown that the Fisher information matrix

$$\boldsymbol{F}_{\boldsymbol{\theta}} = \mathbb{E}_{p(\boldsymbol{y})}\left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y})^T\right] \tag{2.21}$$

can be used to approximate the KL divergence between $p_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\boldsymbol{y})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{y})$ for sufficiently small $\delta\boldsymbol{\theta}$, i.e.,

$$\mathrm{KL}\left(p_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\boldsymbol{y}) || p_{\boldsymbol{\theta}}(\boldsymbol{y})\right) \approx \delta\boldsymbol{\theta}^T \boldsymbol{F}_{\boldsymbol{\theta}} \delta\boldsymbol{\theta}. \tag{2.22}$$

The natural gradient update $\delta\boldsymbol{\theta}^{\mathrm{NG}}$ is now defined as the update $\delta\boldsymbol{\theta}$ that is the most similar to the traditional 'vanilla' gradient $\delta\boldsymbol{\theta}^{\mathrm{VG}}$ update that has a bounded distance

$$\mathrm{KL}(p_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\boldsymbol{y}) || p_{\boldsymbol{\theta}}(\boldsymbol{y})) \leq \epsilon$$

in the distribution space. Hence, we can formulate the following optimization program

$$\delta\boldsymbol{\theta}^{NG} = \mathrm{argmax}_{\delta\boldsymbol{\theta}}\, \delta\boldsymbol{\theta}^T \delta\boldsymbol{\theta}^{\mathrm{VG}} \quad \text{s.t.} \quad \delta\boldsymbol{\theta}^T \mathbf{F}_{\boldsymbol{\theta}} \delta\boldsymbol{\theta} \leq \varepsilon. \tag{2.23}$$

The solution of this program is given by $\delta\boldsymbol{\theta}^{\mathrm{NG}} \propto \boldsymbol{F}_{\boldsymbol{\theta}}^{-1} \delta\boldsymbol{\theta}^{\mathrm{VG}}$ up to a scaling factor. The proportionality factor for the update step is typically subsumed into the learning rate. The natural gradient linearly transforms the traditional gradient by the inverse Fisher matrix, which renders the parameter update also invariant to linear transformations of the parameters of the distribution, i.e., if two parametrizations have the same representative power, the natural gradient update will be identical. As the Fisher information matrix is always positive definite, the natural gradient always rotates the traditional gradient by less than 90 degrees, and, hence, all convergence guarantees from standard gradient-based optimization remain. In contrast to the traditional gradient, the natural gradient avoids premature convergence on plateaus and overaggressive steps on steep ridges due to its isotropic convergence properties [3, 78].

Natural Policy Gradients. The intuition of the natural gradients to limit the distance between two subsequent distributions is also useful for policy search. Here, we want to maintain a limited step-width in the trajectory distribution space, i.e.,

$$\mathrm{KL}(p_{\boldsymbol{\theta}}(\boldsymbol{\tau})||p_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\boldsymbol{\tau})) \approx \delta\boldsymbol{\theta}^T \boldsymbol{F}_{\boldsymbol{\theta}} \delta\boldsymbol{\theta} \leq \epsilon\,.$$

The Fisher information matrix

$$\boldsymbol{F}_{\boldsymbol{\theta}} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})^T\right]$$

is now computed for the trajectory distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$. The natural policy gradient $\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}} J_{\boldsymbol{\theta}}$ is therefore given by

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}} J_{\boldsymbol{\theta}} = \boldsymbol{F}_{\boldsymbol{\theta}}^{-1} \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}, \qquad (2.24)$$

where $\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}$ can be estimated by any traditional policy gradient method.

The difference between the natural and traditional policy gradient for learning a simple linear feedback policy is shown in Figure 2.2. In this example, a scalar controller gain and the variance of the policy are optimized. While the traditional gradient quickly reduces the variance of the policy, and, hence will stop exploring, the natural gradient only gradually decreases the variance, and, in the end, finds the optimal solution faster.

Step-Based Natural Gradient Methods

Similar to the gradient, the Fisher information matrix can also be decomposed in the policy derivatives of the single time steps [8]. In [62, 61], it was shown that the Fisher information matrix of the trajectory distribution can be written as the average Fisher information matrices for each time step, i.e.,

$$\boldsymbol{F}_{\boldsymbol{\theta}} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)^T\right]. \qquad (2.25)$$

Consequently, as it is the case for estimating the policy gradient, the transition model is not needed for estimating $\boldsymbol{F}_{\boldsymbol{\theta}}$. Still, estimating $\boldsymbol{F}_{\boldsymbol{\theta}}$

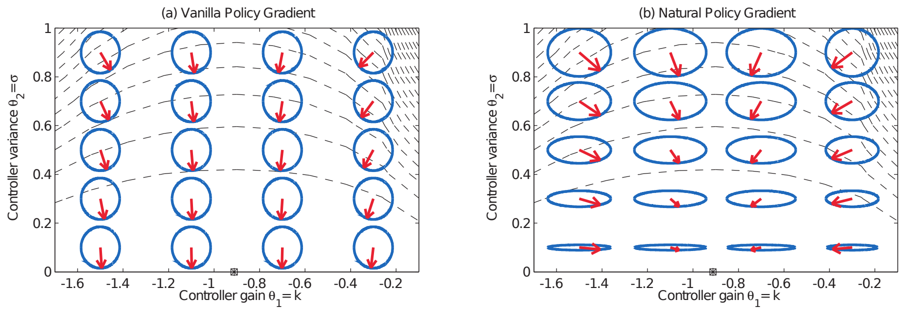


Fig. 2.2 Comparison of the natural gradient to the traditional gradient on a simple linear task with quadratic cost function. The controller has two parameters, the feedback gain k and the variance σ^2. The main difference between the two methods is how the change in parameters is punished, i.e., the distance between current and next policy parameters. This distance is indicated by the blue ellipses in the contour plot while the dashed lines show the expected return. The red arrows indicate the resulting gradient. While the traditional gradient quickly reduces the variance of the policy, the natural gradient only gradually decreases the variance, and therefore continues to explore.

from samples can be difficult, as $\boldsymbol{F}_{\boldsymbol{\theta}}$ contains $O(d^2)$ parameters, where d is the dimensionality of $\boldsymbol{\theta}$. However, the Fisher information matrix $\boldsymbol{F}_{\boldsymbol{\theta}}$ does not need to be estimated explicitly if we use *compatible function approximations*, which we will introduce in the next paragraph.

Compatible Function Approximation. In the PGT algorithm, given in Section 2.4.1.2, the policy gradient was given by

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{PG}} J_{\boldsymbol{\theta}} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)\left(Q_t^{\pi}(\boldsymbol{x}_t,\boldsymbol{u}_t) - b_t(\boldsymbol{x}_t)\right)\right]. \quad (2.26)$$

Instead of using the future rewards of a single roll-out to estimate $Q_t^{\pi}(\boldsymbol{x}_t,\boldsymbol{u}_t) - b_t(\boldsymbol{x})$, we can also use function approximation [79] to estimate the value, i.e., estimate a function $\tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}_t,\boldsymbol{u}_t,t) = \boldsymbol{\psi}_t(\boldsymbol{x}_t,\boldsymbol{u}_t)^T\boldsymbol{w}$ such that $\tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}_t,\boldsymbol{u}_t,t) \approx Q_t^{\pi}(\boldsymbol{x}_t,\boldsymbol{u}_t) - b_t(\boldsymbol{x})$. The quality of the approximation is determined by the choice of the basis functions $\psi_t(\boldsymbol{x}_t,\boldsymbol{u}_t)$, which might explicitly depend on the time step t. A good function approximation does not change the gradient in expectation, i.e., it does not introduce a bias. To find basis functions $\psi_t(\boldsymbol{x}_t,\boldsymbol{u}_t)$ that fulfill this condition, we will first assume that we already found a parameter vector $\boldsymbol{w}$ which approximates Q_t^π. For simplicity, we will for now assume that the baseline $b_t(\boldsymbol{x})$ is zero. A parameter vector $\boldsymbol{w}$, which approximates Q_t^π, also minimizes the squared approximation error. Thus, $\boldsymbol{w}$ has to satisfy

$$\frac{\partial}{\partial \boldsymbol{w}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left[\sum_{t=0}^{T-1} \left(Q_t(\boldsymbol{x}_t, \boldsymbol{u}_t) - \tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}_t, \boldsymbol{u}_t, t) \right)^2 \right] = 0. \tag{2.27}$$

Computing the derivative yields

$$2\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left[\sum_{t=0}^{T-1} (Q_t(\boldsymbol{x}_t, \boldsymbol{u}_t) - \tilde{A}_w(\boldsymbol{x}_t, \boldsymbol{u}_t, t)) \frac{\partial}{\partial \boldsymbol{w}} \tilde{A}_w(\boldsymbol{x}_t, \boldsymbol{u}_t, t) \right] = 0 \tag{2.28}$$

with $\partial \tilde{A}_w(\boldsymbol{x}_t, \boldsymbol{u}_t, t)/\partial \boldsymbol{w} = \boldsymbol{\psi}_t(\boldsymbol{x}_t, \boldsymbol{u}_t)$. By subtracting this equation from the Policy Gradient Theorem in Equation (2.18), it is easy to see that

$$\nabla_{\mathrm{PG}} J(\boldsymbol{\theta}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left[\sum\nolimits_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t | \boldsymbol{x}_t, t) \tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}_t, \boldsymbol{u}_t, t) \right] \tag{2.29}$$

if we use $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t | \boldsymbol{x}_t, t)$ as basis functions $\psi_t(\boldsymbol{x}_t, \boldsymbol{u}_t)$ for $\tilde{A}_{\boldsymbol{w}}$.

Using $\boldsymbol{\psi}_t(\boldsymbol{x}_t, \boldsymbol{u}_t) = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t | \boldsymbol{x}_t, t)$ as basis functions is also called *compatible function approximation* [79], as the function approximation is compatible with the policy parametrization. The policy gradient using compatible function approximation can now be written as

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{FA}} J_{\boldsymbol{\theta}} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t | \boldsymbol{x}_t, t) \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t | \boldsymbol{x}_t, t)^T \right] \boldsymbol{w} = \boldsymbol{G}_{\boldsymbol{\theta}} \boldsymbol{w}. \tag{2.30}$$

Hence, in order to compute the traditional gradient, we have to estimate the weight parameters $\boldsymbol{w}$ of the advantage function and the matrix $\boldsymbol{G}_{\boldsymbol{\theta}}$. However, as we will see in the next section, the matrix $\boldsymbol{G}_{\boldsymbol{\theta}}$ cancels out for the natural gradient, and, hence, computing the natural gradient reduces to computing the weights $\boldsymbol{w}$ for the compatible function approximation.

Step-Based Natural Policy Gradient. The result given in Equation (2.30) implies that the policy gradient $\nabla_{\boldsymbol{\theta}}^{\mathrm{FA}} J_{\boldsymbol{\theta}}$ using the compatible function approximation already contains the Fisher information matrix as $\boldsymbol{G}_{\boldsymbol{\theta}} = \boldsymbol{F}_{\boldsymbol{\theta}}$. Hence, the calculation of the step-based natural gradient simplifies to

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}} J_{\boldsymbol{\theta}} = \boldsymbol{F}_{\boldsymbol{\theta}}^{-1} \nabla_{\boldsymbol{\theta}}^{\mathrm{FA}} J_{\boldsymbol{\theta}} = \boldsymbol{w}. \tag{2.31}$$

The natural gradient still requires estimating the function $\tilde{A}_{\boldsymbol{w}}$. Due to the baseline $b_t(\boldsymbol{x})$, the function $\tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}_t, \boldsymbol{u}_t, t)$ can be interpreted as the advantage function, i.e., $\tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}_t, \boldsymbol{u}_t, t) \approx Q_t^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t) - V_t(\boldsymbol{x}_t)$. We can check that $\tilde{A}_{\boldsymbol{w}}$ is an advantage function by observing that $\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}_t, \boldsymbol{u}_t, t)\right] = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)\right] \boldsymbol{w} = 0$. The advantage function $\tilde{A}_{\boldsymbol{w}}$ can be estimated by using temporal difference methods [80, 13]. However, in order to estimate the advantage function, such methods also require an estimate of the value function $V_t(\boldsymbol{x}_t)$ [61]. While the advantage function would be easy to learn as its basis functions are given by the compatible function approximation, appropriate basis functions for the value function are typically more difficult to specify. Hence, we typically want to find algorithms which avoid estimating a value function.

Episodic Natural Actor Critic. One such algorithm is the episodic Natural Actor Critic (eNAC) algorithm [60]. In the episodic policy search setup, i.e., with a limited time-horizon T, the estimation of the value function V_t can be avoided by considering whole sample paths. To see this, we first rewrite the Bellman equation for the advantage function

$$\tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{u}, t) + V_t(\boldsymbol{x}) = r_t(\boldsymbol{x}, \boldsymbol{u}) + \int p(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{u}) V_t(\boldsymbol{x}') d\boldsymbol{x}', \tag{2.32}$$

where $V_T(\boldsymbol{x}) = r_T(\boldsymbol{x})$ is the reward for the final state. Equation (2.32) can be rewritten as

$$\tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}_t, \boldsymbol{u}_t, t) + V_t(\boldsymbol{x}_t) = r_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + V_t(\boldsymbol{x}_{t+1}) + \epsilon, \tag{2.33}$$

for a single transition from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$, where ϵ is a zero-mean noise term. We now sum up Equation (2.33) along a sample path and get the following condition

$$\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)\boldsymbol{w} + V_0(\boldsymbol{x}_0) = \sum_{t=0}^{T-1} r_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + r_T(\boldsymbol{x}_T) \tag{2.34}$$

Algorithm 8 Episodic Natural Actor Critic

```
Input: Policy parametrization $\boldsymbol{\theta}$,
        data-set $\mathcal{D} = \left\{\boldsymbol{x}_{1:T}^{[i]}, \boldsymbol{u}_{1:T-1}^{[i]}, r_{1:T}^{[i]}\right\}_{i=1\ldots N}$
for each sample $i = 1 \ldots N$ do
    Compute returns: $R^{[i]} = \sum_{t=0}^{T} r_t^{[i]}$
    Compute features: $\boldsymbol{\psi}^{[i]} = \begin{bmatrix} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(\boldsymbol{u}_t^{[i]} \middle| \boldsymbol{x}_t^{[i]}, t\right) \\ \boldsymbol{\varphi}(\boldsymbol{x}_0^{[i]}) \end{bmatrix}$
end for
Fit advantage function and initial value function
    $\boldsymbol{R} = \left[R^{[1]}, \ldots, R^{[N]}\right]^T, \quad \boldsymbol{\Psi} = \left[\boldsymbol{\psi}^{[1]}, \ldots, \boldsymbol{\psi}^{[N]}\right]^T$
    $\begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{v} \end{bmatrix} = (\boldsymbol{\Psi}^T\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^T\boldsymbol{R}$
return $\nabla_{\boldsymbol{\theta}}^{\text{eNAC}} J_{\boldsymbol{\theta}} = \boldsymbol{w}$
```

with $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)\boldsymbol{w} = \tilde{A}_{\boldsymbol{w}}(\boldsymbol{x}_t, \boldsymbol{u}_t, t)$. Now, the value function V_t needs to be estimated only for the first time step. For a single start state $\boldsymbol{x}_0$, estimating $V_0(\boldsymbol{x}_0) = v_0$ reduces to estimating a constant v_0. For multiple start states $\boldsymbol{x}_0$, $\boldsymbol{V_t}$ needs to be parametrized $V_0(\boldsymbol{x}_0) \approx \tilde{V}_{\boldsymbol{v}}(\boldsymbol{x}_0) = \boldsymbol{\varphi}(\boldsymbol{x})^T\boldsymbol{v}$. By using multiple sample paths $\boldsymbol{\tau}^{[i]}$, we get a regression problem whose solution is given by

$$\begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{v} \end{bmatrix} = (\boldsymbol{\Psi}^T\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^T\boldsymbol{R}, \tag{2.35}$$

where the matrix $\boldsymbol{\Psi}$ contains the policy and value function features of the sample paths, i.e.,

$$\boldsymbol{\psi}^{[i]} = \left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(\boldsymbol{u}_t^{[i]} \middle| \boldsymbol{x}_t^{[i]}, t\right), \boldsymbol{\varphi}(\boldsymbol{x}^{[i]})^T\right], \quad \boldsymbol{\Psi} = \left[\boldsymbol{\psi}^{[1]}, \ldots, \boldsymbol{\psi}^{[N]}\right]^T$$

and $\boldsymbol{R}$ contains the returns of the sample paths. The eNAC algorithm is illustrated in Algorithm 8.

Natural Actor Critic. While the eNAC algorithm is efficient for the episodic reinforcement learning formulation, it uses the returns

$R^{[i]}$ for evaluating the policy, and consequently, gets less accurate for large time-horizons due to the large variance of the returns. For learning problems with a large time horizon, especially, for infinite horizon tasks, the convergence speed can be improved by directly estimating the value function V_t. Such a strategy is implemented by the Natural Actor Critic Algorithm (NAC) algorithm [61]. The NAC algorithm estimates the advantage function and the value function by applying temporal difference methods [80, 13]. To do so, temporal difference methods have to first be adapted to learn the advantage function.

We start this derivation by first writing down the Bellman equation in terms of the advantage function in the infinite horizon formulation

$$Q^{\pi}(\boldsymbol{x},\boldsymbol{u}) = A^{\pi}(\boldsymbol{x},\boldsymbol{u}) + V^{\pi}(\boldsymbol{x}) = r(\boldsymbol{x},\boldsymbol{u}) + \gamma \int p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u}) V^{\pi}(\boldsymbol{x}') d\boldsymbol{x}\,. \tag{2.36}$$

Note that we now discuss the infinite horizon case, i.e., all functions are time independent and we introduced the discount factor γ. By inserting $\tilde{A}(\boldsymbol{x},\boldsymbol{u}) \approx \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})\boldsymbol{w}$ and $V^{\pi}(\boldsymbol{x}) \approx \boldsymbol{\varphi}(\boldsymbol{x})^T \boldsymbol{v}$, we can rewrite the Bellman equation as a set of linear equations

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(\boldsymbol{u}^{[i]} \middle| \boldsymbol{x}^{[i]}\right) \boldsymbol{w} + \boldsymbol{\varphi}(\boldsymbol{x}^{[i]})^T \boldsymbol{v} = r(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}) + \gamma \boldsymbol{\varphi}(\boldsymbol{x}'^{[i]})^T \boldsymbol{v} + \boldsymbol{\epsilon}. \tag{2.37}$$

One efficient method to estimate $\boldsymbol{w}$ and $\boldsymbol{v}$ is to use the LSTD-Q(λ) [13, 61] algorithm. A simplified version of the NAC algorithm which uses LSTD-Q(0) to estimate $\boldsymbol{w}$ and $\boldsymbol{v}$ is given in Algorithm 9. For a more detailed discussion of the LSTD-Q algorithm we refer to the corresponding papers [13, 61, 42].

Episode-Based Natural Policy Gradients

The beneficial properties of the natural gradient can also be exploited for episode-based algorithms[88, 78]. While such methods come from the area of evolutionary algorithms, as they always maintain a 'population' of parameter-samples, they perform gradient ascent on a fitness function which is in the reinforcement learning context the expected long-term reward $J_{\boldsymbol{\omega}}$ of the upper-level policy. Hence, we categorize these methods as Policy Gradient methods.

Algorithm 9 Natural Actor Critic

```
Input: Policy parametrization $\boldsymbol{\theta}$,
        data-set $\mathcal{D} = \left\{\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}, r^{[i]}, \boldsymbol{x}'^{[i]}\right\}_{i=1\ldots N}$
for each sample $i = 1 \ldots N$ do
    Compute features for current and successor state:
    $$\boldsymbol{\psi}^{[i]} = \begin{bmatrix} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(\boldsymbol{u}^{[i]} \middle| \boldsymbol{x}^{[i]}\right) \\ \boldsymbol{\varphi}(\boldsymbol{x}^{[i]}) \end{bmatrix}, \quad \boldsymbol{\psi}'^{[i]} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{\varphi}(\boldsymbol{x}'^{[i]}) \end{bmatrix}$$
end for
Compute LSTD-Q solution
    $$\boldsymbol{b} = \sum_{i=1}^{N} \boldsymbol{\psi}^{[i]} r^{[i]}, \qquad \boldsymbol{A} = \sum_{i=1}^{N} \boldsymbol{\psi}^{[i]} \left(\boldsymbol{\psi}^{[i]} - \gamma \boldsymbol{\psi}'^{[i]}\right)^T$$
    $$\begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{v} \end{bmatrix} = \boldsymbol{A}^{-1} \boldsymbol{b}$$
return $\nabla_{\boldsymbol{\theta}}^{\mathrm{NAC}} J_{\boldsymbol{\theta}} = \boldsymbol{w}$
```

Existing natural gradient methods in parameter space do not use a compatible function approximation to estimate the natural gradient but directly try to estimate the Fisher information matrix which is subsequently multiplied with the likelihood gradient $\nabla_{\boldsymbol{\omega}}^{\mathrm{PE}} J_{\boldsymbol{\omega}}$ given in Equation (2.19) in parameter space, i.e.,

$$\nabla_{\boldsymbol{\omega}}^{\mathrm{NES}} J_{\boldsymbol{\omega}} = \boldsymbol{F}_{\boldsymbol{\omega}}^{-1} \nabla_{\boldsymbol{\omega}}^{\mathrm{PE}} J_{\boldsymbol{\omega}}. \tag{2.38}$$

The natural gradient for parameter space was first used in the Natural Evolution Strategy [88] where the Fisher information matrix was determined empirically. However, the empirical estimation of the Fisher information matrix is problematic as the matrix may not be invertible due to redundant parameters or sampling errors. In [78], the authors compute the Fisher information matrix in closed form for Gaussian policies in parameter space. The authors also give derivations of an optimal baseline for their method. As these derivations are rather complex we refer to the corresponding paper for both derivations. The NES strategy has also been compared with the PEPG algorithm [68], indicating that NES is more efficient for low-dimensional problems while PEPG has advantages for high-dimensional parameter spaces as the second-order type update of the natural gradient gets more difficult.

2.4.2 Expectation Maximization Policy Search Approaches

Policy gradient methods require the user to specify the learning rate. Setting the learning rate can be problematic and often results in an unstable learning process or slow convergence [38]. This problem can be avoided by formulating policy search as an inference problem with latent variables and, subsequently, using the Expectation-Maximization (EM) algorithm to infer a new policy. As in the standard Expectation-Maximization algorithm, the parameter update is computed as a weighted maximum likelihood estimate which has a closed form solution for most of the used policies. Hence, no learning rate is required.

We will first review the standard EM-algorithm and, subsequently, reformulate policy search as an inference problem by treating the reward as improper probability distribution. Finally, we will explain the resulting EM-based policy search algorithms.

2.4.2.1 The Standard Expectation Maximization Algorithm

The EM-algorithm [46, 12] is a well known algorithm for determining the maximum likelihood solution of a probabilistic latent variable model. Let us assume that $\boldsymbol{y}$ defines an observed random variable, $\boldsymbol{z}$ an unobserved random variable and $p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{z})$ is the parametrized joint distribution of observed and unobserved variables with parameters $\boldsymbol{\theta}$. As $\boldsymbol{z}$ is unobserved, it needs to be marginalized out to compute the likelihood of the parameters, i.e., $p_{\boldsymbol{\theta}}(\boldsymbol{y}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{z}) d\boldsymbol{z}$. Given a data set $\boldsymbol{Y} = [\boldsymbol{y}^{[1]}, \dots \boldsymbol{y}^{[N]}]^T$, we now want to maximize the log-marginal-likelihood

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{Y}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{[i]}) = \sum_{i=1}^{N} \log \int p_{\boldsymbol{\theta}}(\boldsymbol{y}^{[i]}, \boldsymbol{z}) d\boldsymbol{z} \tag{2.39}$$

with respect to the parameters $\boldsymbol{\theta}$, where we assumed i.i.d. data-points, i.e., $p_{\boldsymbol{\theta}}(\boldsymbol{Y}) = \prod_i p_{\boldsymbol{\theta}}(\boldsymbol{y}^{[i]})$. Since the logarithm is acting on the marginal distribution $\int p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{z}) d\boldsymbol{z}$ instead of the joint distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{z})$, we can not obtain a closed form solution for the parameters $\boldsymbol{\theta}$ of our probability model $p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{z})$.

The EM-algorithm is an iterative procedure for estimating the maximum likelihood solution of latent variable models where the parameter updates of every iteration can be obtained in closed form. We will closely follow the derivation of EM from [12] as it can directly be applied to the policy search setup.

The costly marginalization over the hidden variables can be avoided by introducing an auxiliary distribution $q(\boldsymbol{Z})$, which we will denote as variational distribution, that is used to decompose the marginal log-likelihood by using the identity $p_{\boldsymbol{\theta}}(\boldsymbol{Y}) = p_{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{Z}) / p_{\boldsymbol{\theta}}(\boldsymbol{Z}|\boldsymbol{Y})$, i.e.,

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{Y}) &= \int q(\boldsymbol{Z}) \log p_{\boldsymbol{\theta}}(\boldsymbol{Y}) d\boldsymbol{Z} = \int q(\boldsymbol{Z}) \log \frac{q(\boldsymbol{Z}) p_{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{Z})}{q(\boldsymbol{Z}) p_{\boldsymbol{\theta}}(\boldsymbol{Z}|\boldsymbol{Y})} d\boldsymbol{Z} \\
&= \int q(\boldsymbol{Z}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{Z})}{q(\boldsymbol{Z})} d\boldsymbol{Z} - \int q(\boldsymbol{Z}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{Z}|\boldsymbol{Y})}{q(\boldsymbol{Z})} d\boldsymbol{Z} \\
&= \mathcal{L}_{\boldsymbol{\theta}}(q) + \mathrm{KL}\left(q(\boldsymbol{Z}) || p_{\boldsymbol{\theta}}(\boldsymbol{Z}|\boldsymbol{Y})\right). \qquad (2.40)
\end{aligned}
$$

Since the KL-divergence is always larger or equal to zero, the term $\mathcal{L}_{\boldsymbol{\theta}}(q)$ is a lower bound of the log marginal-likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{Y})$. The two update steps in EM each correspond to maximizing the lower bound $\mathcal{L}$ and minimizing the KL-divergence term.

Expectation Step. In the expectation step (E-step), we update the variational distribution $q(\boldsymbol{Z})$ by minimizing the KL-divergence $\mathrm{KL}\left(q(\boldsymbol{Z}) || p_{\boldsymbol{\theta}}(\boldsymbol{Z}|\boldsymbol{Y})\right)$ which is equivalent to setting $q(\boldsymbol{Z}) = p_{\boldsymbol{\theta}}(\boldsymbol{Z}|\boldsymbol{Y})$. Note that the lower bound $\mathcal{L}_{\boldsymbol{\theta}}(q)$ is tight after each E-step, i.e., $\log p_{\boldsymbol{\theta}}(\boldsymbol{Y}) = \mathcal{L}_{\boldsymbol{\theta}}(q)$, as the KL-divergence $\mathrm{KL}\left(q(\boldsymbol{Z}) || p_{\boldsymbol{\theta}}(\boldsymbol{Z}|\boldsymbol{Y})\right)$ has been set to zero by the E-step. As $\log p_{\boldsymbol{\theta}}(\boldsymbol{Y})$ is unaffected by the change of $q(\boldsymbol{Z})$, we observe from Equation (2.40) that the lower bound $\mathcal{L}_{\boldsymbol{\theta}}(q)$ has to increase if we decrease the KL-divergence.

Maximization Step. In the maximization step (M-step), we optimize the lower bound with respect to $\boldsymbol{\theta}$, i.e.,

$$\begin{aligned}\boldsymbol{\theta}_{\text{new}} &= \text{argmax}_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}}(q) = \text{argmax}_{\boldsymbol{\theta}} \int q(\boldsymbol{Z}) \log p_{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{Z}) d\boldsymbol{Z} + H(q) \\ &= \text{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{q(\boldsymbol{Z})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{Z})\right] = \text{argmax}_{\boldsymbol{\theta}} \mathcal{Q}_{\boldsymbol{\theta}}(q), \qquad (2.41)\end{aligned}$$

where the term $H(q)$ denotes the entropy of q and can be neglected for estimating $\boldsymbol{\theta}_{\text{new}}$. The term in Equation (2.41) is also denoted as the *expected complete data log-likelihood* $\mathcal{Q}_{\boldsymbol{\theta}}(q)$. The log now directly acts on the joint distribution $p_{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{Z})$, and, hence, the M-step can be obtained in closed form. By examining the expected complete data log-likelihood

$$\begin{aligned}\mathcal{Q}_{\boldsymbol{\theta}}(q) &= \int q(\boldsymbol{Z}) \log p_{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{Z}) d\boldsymbol{Z} \qquad (2.42) \\ &= \sum_{i=1}^{N} \int q_i(\boldsymbol{z}) \log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{[i]}, \boldsymbol{z}) d\boldsymbol{Z} \qquad (2.43)\end{aligned}$$

in more detail, we can see that the M-step is based on a *weighted maximum likelihood estimate* of $\boldsymbol{\theta}$ using the complete data points $[\boldsymbol{y}^{[i]}, \boldsymbol{z}]$ weighted by $q_i(\boldsymbol{z})$.

Note that after the E-step, the KL-term of Equation (2.40) is set to zero and, hence, the KL-term can only increase in the M-step. Consequently, $\log p_{\boldsymbol{\theta}}(\boldsymbol{Y})$ is increased even more than the lower bound $\mathcal{L}$. The EM-algorithm is guaranteed to converge to a local maximum of the marginal log-likelihood $\log p_{\theta}(\boldsymbol{Y})$ as the lower bound is increased in each E-step and M-step, and the lower bound is tight after each E-step.

2.4.2.2 Policy Search as an Inference Problem

We will first formulate policy search as a latent variable inference problem and then show how EM can be applied to solve this problem. To do so, we define a binary reward event R as our observed variable. As we want to maximize the reward, we always want to observe the reward event, i.e., $R = 1$. The probability of this reward event is given by $p(R = 1|\boldsymbol{\tau})$. The trajectories $\boldsymbol{\tau}$ are the latent variables in our model. A graphical model of policy search formulated as an inference problem is given in Figure 2.3.

Maximizing the reward implies maximizing the probability of the reward event, and, hence, our trajectory distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ needs to assign high probability to trajectories with high reward probability $p(R=1|\boldsymbol{\tau})$. As we are only concerned with the case $R=1$ for estimating the trajectory distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$, we will write $p(R|\boldsymbol{\tau})$ instead of $p(R=1|\boldsymbol{\tau})$.

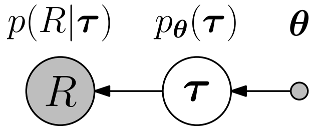


Fig. 2.3 Graphical Model for inference-based policy search. We introduce a binary reward event $R=1$ as observation, the latent variables are given by the trajectories $\boldsymbol{\tau}$. We want to find the maximum likelihood solution for the parameters $\boldsymbol{\theta}$ of observing the reward, i.e., $\boldsymbol{\theta}_{\text{new}} = \operatorname{argmax}_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(R)$.

If the return $R(\boldsymbol{\tau})$ of a trajectory is bounded, it can be directly transformed into a non-normalized probability distribution, i.e., $p(R|\boldsymbol{\tau}) \propto R(\boldsymbol{\tau}) - \min_{\boldsymbol{\tau}_j} R(\boldsymbol{\tau}_j)$ [19]. Otherwise, an exponential transformation of the reward signal can be used [59, 84], i.e., $p(R|\boldsymbol{\tau}) \propto \exp\left(\beta R(\boldsymbol{\tau})\right)$. This exponential transformation implies a soft-max distribution for the trajectories conditioned on the observation of the reward event, i.e.,

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|R) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \exp(\beta R(\boldsymbol{\tau}))}{\int p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \exp(\beta R(\boldsymbol{\tau})) d\boldsymbol{\tau}}.$$

The parameter β denotes the inverse temperature of the soft-max distribution. The higher we choose β, the more greedy the policy update becomes. This parameter is either specified by the user [38] or can be set by heuristics, for example, setting β to a multiple of the standard deviation of the rewards [48, 49].

We want to find a parameter vector $\boldsymbol{\theta}$ that maximizes the probability of the reward event. In other words, we want to find the maximum likelihood solution for the log marginal-likelihood

$$\log p_{\boldsymbol{\theta}}(R) = \log \int_{\boldsymbol{\tau}} p(R|\boldsymbol{\tau}) p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) d\boldsymbol{\tau}. \tag{2.44}$$

As for the standard EM algorithm, a variational distribution $q(\boldsymbol{\tau})$ is used to decompose the log-marginal likelihood into two terms

$$\log p_{\boldsymbol{\theta}}(R) = \mathcal{L}_{\boldsymbol{\theta}}(q) + \mathrm{KL}\left(q(\boldsymbol{\tau}) || p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|R)\right), \tag{2.45}$$

where $p(\boldsymbol{\tau}|R)$ is denoted as the *reward-weighted* trajectory distribution

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|R) = \frac{p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}{\int p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}}(\boldsymbol{\tau})d\boldsymbol{\tau}} \propto p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}}(\boldsymbol{\tau}). \tag{2.46}$$

In the E-step, we minimize $\mathrm{KL}\left(q(\boldsymbol{\tau})||p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|R)\right)$, and, hence, set $q(\boldsymbol{\tau})$ to the reward weighted model distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|R)$. In the M-step, we maximize the expected complete data log-likelihood

$$\begin{aligned}
\mathrm{argmax}_{\boldsymbol{\theta}} \mathcal{Q}_{\boldsymbol{\theta}}(q) &= \mathrm{argmax}_{\boldsymbol{\theta}} \int q(\boldsymbol{\tau}) \log\big(p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\big)d\boldsymbol{\tau} \\
&= \mathrm{argmax}_{\boldsymbol{\theta}} \int q(\boldsymbol{\tau}) \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})d\boldsymbol{\tau} + f(q) \\
&= \mathrm{argmin}_{\boldsymbol{\theta}} \mathrm{KL}\left(q(\boldsymbol{\tau})||p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\right). \qquad (2.47)
\end{aligned}$$

Hence, the E- and the M-step use different KL-divergences for their iterative updates. We distinguish between two EM update procedures, called Monte-Carlo (MC-)EM approaches [38, 59] and Variational Inference for Policy Search [48], which we will discuss in the following sections. Both update procedures use different approximations to minimize the KL-divergences.

2.4.2.3 Monte-Carlo EM-based Policy Search.

Some of the most efficient policy search methods are Monte-Carlo Expectation-Maximization (MC-EM) methods [38, 59, 38, 85]. The MC-EM algorithm [46] is a variant of EM that uses a sample based approximation for the variational distribution q, i.e., in the E-step, MC-EM minimizes the KL-divergence $\mathrm{KL}(q(\boldsymbol{Z})||p_{\boldsymbol{\theta}}(\boldsymbol{Z}|\boldsymbol{Y})$ by using samples $Z_j \sim p_{\boldsymbol{\theta}}(\boldsymbol{Z}|\boldsymbol{Y})$. Subsequently, these samples $\boldsymbol{Z}_j$ are used to estimate the expectation of the complete data log-likelihood

$$\mathcal{Q}_{\boldsymbol{\theta}}(q) = \sum_{j=1}^{K} \log p_{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{Z}_j). \tag{2.48}$$

In terms of policy search, MC-EM methods use samples $\boldsymbol{\tau}^{[i]}$ from the old trajectory distribution $p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})$, where $\boldsymbol{\theta}'$ represents the old policy parameters, to represent the variational distribution $q(\boldsymbol{\tau}) \propto p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})$ over trajectories. As $\boldsymbol{\tau}^{[i]}$ has already been sampled from $p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})$, $p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})$

Algorithm 10 Episode-Based MC-EM Policy Updates

```
Input: inverse temperature $\beta$
        data-set $\mathcal{D}_{\text{ep}} = \left\{ \boldsymbol{s}^{[i]}, \boldsymbol{\theta}^{[i]}, R^{[i]} \right\}_{i=1\ldots N}$
Compute weighting $d^{[i]} = f(R^{[i]})$ for each sample $i$
        e.g., $d^{[i]} \propto \exp(\beta R^{[i]})$
Compute weighted ML solution, see Equation (2.50) and (2.51)
        $\boldsymbol{\omega}_{\text{new}} = \text{argmax}_{\boldsymbol{\omega}} \sum_{i=1}^{N} \sum_{t=0}^{T-1} d^{[i]} \log \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}^{[i]} | \boldsymbol{s}^{[i]})$
    return $\boldsymbol{\omega}_{\text{new}}$
```

cancels with the importance weights and can be skipped from the distribution $q(\boldsymbol{\tau}^{[i]})$ and, hence, $q(\boldsymbol{\tau}^{[i]}) \propto p(R|\boldsymbol{\tau}^{[i]})$. These samples are then used in the M-step for estimating the complete-data log-likelihood. Consequently, in the M-step, we have to maximize

$$\mathcal{Q}_{\boldsymbol{\theta}}(\boldsymbol{\theta}') = \sum_{\boldsymbol{\tau}^{[i]} \sim p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})} p(R|\boldsymbol{\tau}^{[i]}) \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau}^{[i]}) \tag{2.49}$$

with respect to the new policy parameters $\boldsymbol{\theta}$. This maximization corresponds to a weighted maximum likelihood estimate of $\boldsymbol{\theta}$ where each sample $\boldsymbol{\tau}^{[i]}$ is weighted by $d^{[i]} = p(R|\boldsymbol{\tau}^{[i]})$.

Episode-based EM-Algorithms

The episode-based version of MC-EM policy search algorithms can straightforwardly be derived by replacing the trajectory distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau}^{[i]})$ by the upper-level policy $\pi_{\omega}(\boldsymbol{\theta})$ and has been used to generalize the upper-level policy to multiply contexts, i.e., learn $\pi_{\omega}(\boldsymbol{\theta}|\boldsymbol{s})$ [37]. The policy update is given by the weighted maximum likelihood estimate of the parameters $\boldsymbol{\omega}$ of the upper level policy. The general setup for Episode-Based EM-updates is given in Algorithm 10.

Reward Weighted Regression. Reward Weighted Regression (RWR) uses a linear policy for $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$, and, hence, the weighted maximum likelihood estimate performed by the EM-update is given by a weighted linear regression. RWR was introduced in [59] to learn an inverse dynamics model for operational space control. However, the algorithm straightforwardly generalizes to episode-based policy search with multiple contexts. The policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) = \mathcal{N}\left(\boldsymbol{\theta}|\boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{s}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}\right)$ is represented as Gaussian linear model. Given the data-set $\mathcal{D}_{\text{ep}}$ and the weightings $d^{[i]}$ for each sample $(\boldsymbol{s}^{[i]}, \boldsymbol{\theta}^{[i]})$ in $\mathcal{D}_{\text{ep}}$, the weighted maximum likelihood solution for $\boldsymbol{W}$ is given by

$$\boldsymbol{W} = (\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Theta}, \tag{2.50}$$

where λ is a ridge factor, $\boldsymbol{\Phi} = \left[\boldsymbol{\phi}(\boldsymbol{s}^{[1]}), \ldots, \boldsymbol{\phi}(\boldsymbol{s}^{[N]})\right]$ contains the feature vectors of the contexts, the diagonal matrix $\boldsymbol{D}$ contains the weights $d^{[i]}$, and $\boldsymbol{\Theta} = \left[\boldsymbol{\theta}^{[1]}, \ldots, \boldsymbol{\theta}^{[N]}\right]^T$ the parameter vectors $\boldsymbol{\theta}^{[i]}$. The covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ can be updated according to a weighted maximum likelihood estimate. The update equations for $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ are given in Appendix B.

Cost-Regularized Kernel Regression. Cost-Regularized Kernel Regression (CRKR) is the kernelized version of Reward-Weighted-Regression, and was one of the first algorithms used to learn upper-level policies for multiple contexts [38]. Similar to most kernel-regression methods, CRKR uses individual regressions for the individual output dimensions. The policy in CRKR $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}_{\boldsymbol{\omega}}(\boldsymbol{s}), \text{diag}(\boldsymbol{\sigma}_{\boldsymbol{\omega}}(\boldsymbol{s}))$ is modeled as a Gaussian process. The mean and the variance for the h-th output dimension are therefore given by

$$\begin{aligned} \mu_h(\boldsymbol{s}) &= \boldsymbol{k}(\boldsymbol{s})(\boldsymbol{K} + \lambda\boldsymbol{C})^{-1}\boldsymbol{\Theta}_h, & (2.51)\\ \sigma_h^2(\boldsymbol{s}) &= k(\boldsymbol{s}, \boldsymbol{s}) + \lambda - \boldsymbol{k}(\boldsymbol{s})^T(\boldsymbol{K} + \lambda\boldsymbol{C})^{-1}\boldsymbol{k}(\boldsymbol{s}). & (2.52)\end{aligned}$$

The term $\boldsymbol{K} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$ denotes the kernel matrix and $\boldsymbol{k}(\boldsymbol{s}) = \boldsymbol{\phi}(\boldsymbol{s})^T\boldsymbol{\Phi}$ represents the kernel vector for a new context query point $\boldsymbol{s}$, where the feature matrix $\boldsymbol{\Phi} = \left[\boldsymbol{\phi}(\boldsymbol{s}^{[1]}), \ldots, \boldsymbol{\phi}(\boldsymbol{s}^{[N]})\right]$ contains the feature vectors of the contexts. The matrix $\boldsymbol{C}$ is denoted as cost matrix because it is inversely related to the reward weighting used in RWR, i.e., $\boldsymbol{C} = \boldsymbol{D}^{-1}$. The cost matrix is treated as an input dependent noise prior for the Gaussian process [37].

As the standard kernel-regression formulation (for example, see Bishop [12], chapter 6.1), CRKR can be derived from linear regression using the Woodbury Identity [37]. Instead of standard linear regression, reward weighted regression is used to derive CRKR.

As CRKR is a non-parametric method, it does not update a parameter vector. Instead, the policy is determined by the given data set. As CRKR is a kernel method, we do not need to specify a feature vector $\boldsymbol{\phi}(\boldsymbol{s})$, but rather use a kernel. Kernels typically offer more flexibility in modeling a function than user-specified feature vectors. We refer to [65] for more details about kernel-methods for regression. The disadvantage of using a kernel-method is that the output dimensions of the policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ are typically modeled as independent Gaussian distributions, and, hence, no correlations can be modeled. Such uncorrelated exploration strategies might result in a decreased performance of the algorithm as we will discuss in Section 2.1.

Step-based EM-Algorithms

Step-based EM-Algorithms decompose the complete data log-likelihood $\mathcal{Q}_{\boldsymbol{\theta}}(\theta')$ into the single steps of the episode. We denote the parameter vector $\boldsymbol{\theta}'$ as the parameters of the old policy. We first show that $\mathcal{Q}_{\boldsymbol{\theta}}(\theta')$ is a lower bound of the logarithmic expected return $\log J_{\boldsymbol{\theta}}$ where we will assume that no reward transformation has been used, i.e., $p(R|\boldsymbol{\tau}) \propto R(\boldsymbol{\tau})$,

$$\log J_{\boldsymbol{\theta}} = \log \int p_{\boldsymbol{\theta}}(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau} = \log p_{\boldsymbol{\theta}}(R). \tag{2.53}$$

As we know that $\mathcal{Q}_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$ is a lower bound of $\log p_{\boldsymbol{\theta}}(R)$, we conclude that

$$\log J_{\boldsymbol{\theta}} \geq \mathcal{Q}_{\boldsymbol{\theta}}(\boldsymbol{\theta}') = \mathbb{E}_{p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})}\left[R(\tau)\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\right], \tag{2.54}$$

where the *old* policy parameters $\boldsymbol{\theta}'$ have been used for generating the roll-outs. By differentiating $\mathcal{Q}_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$ with respect to the new policy parameters $\boldsymbol{\theta}$, we get

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{Q}_{\boldsymbol{\theta}}(\boldsymbol{\theta}') &= \mathbb{E}_{p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})}\left[R(\boldsymbol{\tau})\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\right],\\ &= \mathbb{E}_{p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})}\left[R(\boldsymbol{\tau})\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}}\log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t,t)\right].\end{aligned} \tag{2.55}$$

Algorithm 11 Step-Based MC-EM Policy Updates

```
Input: Policy parametrization $\boldsymbol{\theta}$
    data-set $\mathcal{D} = \left\{\boldsymbol{x}_{1:T}^{[i]}, \boldsymbol{u}_{1:T-1}^{[i]}, Q_t^{[i]}\right\}_{i=1\ldots N}$
Compute weighting $d_t^{[i]} \propto Q_t^{[i]}$ or $d_t^{[i]} \propto \exp(\beta Q_t^{[i]})$
Compute weighted maximum ML estimate, see Equations (2.58) and (2.59)
    $\boldsymbol{\theta}_{\text{new}} = \text{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{N} \sum_{t=0}^{T-1} d_t^{[i]} \log \pi_{\boldsymbol{\theta}} \left(\boldsymbol{u}_t^{[i]} \middle| \boldsymbol{x}_t^{[i]}, t\right)$
```

Using the same insight as we used for the policy gradient theorem, i.e., past rewards are independent of future actions, we obtain

$$\nabla_{\boldsymbol{\theta}} \mathcal{Q}_{\boldsymbol{\theta}}(\boldsymbol{\theta}') = \mathbb{E}_{p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})} \left[\sum_{t=0}^{T-1} Q_t^{\pi}(\boldsymbol{x}_t, \boldsymbol{u}_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t | \boldsymbol{x}_t, t) \right]. \qquad (2.56)$$

Setting Equation (2.56) to zero corresponds to performing a weighted maximum likelihood estimate on the step-based data-set $\mathcal{D}_{\text{step}}$ for obtaining the new parameters $\boldsymbol{\theta}$ of policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t, t)$. The weighting is in the step-based case given by

$$Q_t^{\pi}(\boldsymbol{x}_t^{[i]}, \boldsymbol{u}_t^{[i]}) \approx \sum_{h=t}^{T-1} r(\boldsymbol{x}_h^{[i]}, \boldsymbol{u}_h^{[i]}).$$

From Equation (2.56) we can see that step-based EM algorithms reduce policy search to an iterative reward-weighted imitation learning procedure. This formulation is used to derive the widely used EM-based policy search algorithm, Policy learning by Weighting Exploration with Returns (PoWER), which was introduced in [38]. The general algorithm for step-based EM algorithms is illustrated in Algorithm 11.

Relation to Policy Gradients. There is a close connection between step-based gradient methods which were introduced in Section 2.4.1 and the step-based EM-based approach. In the limit, if the new parameters $\boldsymbol{\theta}$ are close to the old parameters $\boldsymbol{\theta}'$, we obtain the policy gradient theorem update rule from the lower bound $\mathcal{Q}_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$,

$$\begin{aligned}
\lim_{\boldsymbol{\theta}\to\boldsymbol{\theta}'} \nabla_{\boldsymbol{\theta}} \mathcal{Q}_{\boldsymbol{\theta}}(\boldsymbol{\theta}') &= \lim_{\boldsymbol{\theta}\to\boldsymbol{\theta}'} \mathbb{E}_{p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})}\left[\sum_{t=0}^{T-1} Q_t^{\pi}(\boldsymbol{x}_t,\boldsymbol{u}_t)\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\sum_{t=0}^{T-1} Q_t^{\pi}(\boldsymbol{x}_t,\boldsymbol{u}_t)\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)\right] \\
&= \nabla_{\boldsymbol{\theta}}^{\mathrm{PG}} J_{\boldsymbol{\theta}},
\end{aligned} \tag{2.57}$$

From this comparison we conclude that, unlike policy gradient methods, the EM-based approach allows us to use a different parameter vector $\boldsymbol{\theta}'$ for the expectation of the trajectories than for the estimation of the gradient $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$. The EM-based approach aims at making actions with high future reward $Q_t^{\pi}(\boldsymbol{x},\boldsymbol{u})$ more likely. However, in contrast to policy gradient methods, it neglects the influence of the policy update $\delta\boldsymbol{\theta}$ on the trajectory distribution $p_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\boldsymbol{\tau})$.

However, such a relationship can only be obtained, if we can linearly transform the reward into an improper probability distribution. If we need to use an exponential transformation for the rewards, such a direct relationship with policy gradient methods cannot be established.

Episodic Reward Weighted Regression. Despite the name, Episodic Reward Weighted Regression (eRWR) is the step-based extension of RWR [38], which we presented in the previous section. Similar to RWR, episodic RWR assumes a linear model for the policy, which is in the step-based case given as $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t,t) = \mathcal{N}(\boldsymbol{u}_t|\boldsymbol{W}^T\boldsymbol{\phi}_t(\boldsymbol{x}),\boldsymbol{\Sigma}_{\boldsymbol{u}})$. The weighted maximum likelihood estimate for $\boldsymbol{\theta} = \{\boldsymbol{W},\boldsymbol{\Sigma}_{\boldsymbol{u}}\}$ is now performed on the step-based data set $\mathcal{D}_{\text{step}}$ with inputs $\boldsymbol{x}^{[i]}$, target vectors $\boldsymbol{u}^{[i]}$ and weightings $d^{[i]}$. As we have a Gaussian policy which is linear in the feature vectors $\boldsymbol{\phi}_t$, the weighted ML estimate of the weight vector $\boldsymbol{W}$ is given by a weighted least squares linear regression,

$$\boldsymbol{W}_{\text{new}} = (\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{U}, \tag{2.58}$$

where $\boldsymbol{\Phi} = \left[\boldsymbol{\phi}_0^{[1]},\dots,\boldsymbol{\phi}_{T-1}^{[1]},\dots,\boldsymbol{\phi}_0^{[N]},\dots,\boldsymbol{\phi}_{T-1}^{[N]}\right]$ contains the feature vectors for all time steps t of all trajectories $\boldsymbol{\tau}^{[i]}$, $\boldsymbol{D}$ is the diagonal weighting matrix containing the weightings $d_t^{[i]}$ of each sample and the matrix $\boldsymbol{U}$ contains the control vectors $\boldsymbol{u}_t^{[i]}$ for all t and i. For more details on Equation (2.58) please refer to Appendix B. The update of the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{u}}$ of $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ can also be obtained by a weighted maximum likelihood estimate and is given in Appendix B, see Equation (4.4).

Policy learning by Weighting Exploration with Returns. The policy update of the PoWER algorithm [38] is similar to episodic RWR, however, PoWER uses a more structured exploration strategy which typically results in better performance of PoWER in comparison to episodic RWR. To simplify our discussion we will for now assume that the control action u_t is one dimensional. RWR directly perturbs the controls $u_t = \boldsymbol{w}^T\boldsymbol{\phi}_t + \boldsymbol{\epsilon}_t$ with zero-mean Gaussian noise. Instead, PoWER applies the perturbation to the parameters $\boldsymbol{w}$, i.e. $u_t = (\boldsymbol{w} + \boldsymbol{\epsilon}_t)^T\boldsymbol{\phi}_t$, where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$ is the noise term applied to the parameter vector at time step t. Such a policy can be written as $\pi_t(\boldsymbol{u}_t|\boldsymbol{x}_t) = \mathcal{N}(u_t|\boldsymbol{w}^T\boldsymbol{\phi}_t, \boldsymbol{\phi}_t^T\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\phi}_t)$, i.e., as Gaussian policy where the variance also depends on the current features $\boldsymbol{\Phi}_t$.

If we assume that $\Sigma_{\boldsymbol{w}}$ is known, we can again determine the weighted maximum likelihood solution for the parameters $\boldsymbol{w}$. This solution is given in [38] as

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w}_{\text{old}} + \mathbb{E}\left[\sum_{t=0}^{T-1}\boldsymbol{L}_t(\boldsymbol{x})Q_t^{\pi}(\boldsymbol{x}_t,\boldsymbol{u}_t)\right]^{-1}\mathbb{E}\left[\sum_{t=0}^{T-1}\boldsymbol{L}_t(\boldsymbol{x})Q_t^{\pi}(\boldsymbol{x}_t,\boldsymbol{u}_t)\boldsymbol{\epsilon}_t\right],$$

where $\boldsymbol{L}_t(\boldsymbol{x}) = \boldsymbol{\phi}_t(\boldsymbol{x})\boldsymbol{\phi}_t(\boldsymbol{x})^T(\boldsymbol{\phi}_t(\boldsymbol{x})^T\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\phi}_t(\boldsymbol{x}))^{-1}$. As we can see, the exploration noise $\boldsymbol{\epsilon}_t$ is weighted by the returns Q_t^{π} to obtain the new parameter vector $\boldsymbol{w}$. For the derivation of this equation we refer to [38]. However, the update rule of PoWER can also be written in terms of matrices, where we use the action vectors $\boldsymbol{u}_t$ instead of using the noise terms $\boldsymbol{\epsilon}_t$ as target values, i.e,

$$\boldsymbol{w}_{\text{new}} = (\boldsymbol{\Phi}^T\tilde{\boldsymbol{D}}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\tilde{\boldsymbol{D}}\boldsymbol{U}, \tag{2.59}$$

where $\boldsymbol{U}$ contains the actions of all time steps and all trajectories, $\boldsymbol{\Phi}^T$ is defined as in Equation (2.50) and $\tilde{\boldsymbol{D}} \in \mathbb{R}^{NT\times NT}$ is a diagonal weighting matrix with the entries $\tilde{d}_t^{[i]} = \left(\boldsymbol{\phi}_t\left(\boldsymbol{x}^{[i]}\right)^T\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\phi}_t\left(\boldsymbol{x}^{[i]}\right)\right)^{-}1Q_t^{[i]}$ for each sample i and time step t. For further details please consult Appendix B. We can now see, as the only difference between the eRWR policy and the policy used in PoWER is the state-dependent variance term, the only difference in the policy update is that in PoWER the data-points are additionally weighted by the precision $\left(\boldsymbol{\phi}_t\left(\boldsymbol{x}^{[i]}\right)^T \boldsymbol{\Sigma}_{\boldsymbol{w}} \boldsymbol{\phi}_t\left(\boldsymbol{x}^{[i]}\right)\right)^{-1}$ of the policy for state $\boldsymbol{x}^{[i]}$. Consequently, data-points with less variance have a higher influence on the result of the regression. As this notation does not contain the noise terms $\boldsymbol{\epsilon}_t^{[i]}$, it is also compatible with Algorithm 11.

2.4.2.4 Variational Inference-based Methods

As we have seen, the MC-EM approach uses a weighted maximum likelihood estimate to obtain the new parameters $\boldsymbol{\theta}$ of the policy. While a weighted maximum likelihood estimate can be computed efficiently, such an approach might also suffer from a caveat: it averages over several modes of the reward function. Such a behavior might result in slow convergence to good policies as the average of several modes might be in an area with low reward [48].

Moment Projection and Information Projection. We observe that the maximization used for the MC-EM approach as defined in Equation (2.49) is equivalent to minimizing

$$\mathrm{KL}(p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})\|p_{\boldsymbol{\theta}}(\boldsymbol{\tau})) = \int p(R|\boldsymbol{\tau}^{[i]})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau}^{[i]}) \log \frac{p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau}^{[i]})}{p_{\boldsymbol{\theta}}(\boldsymbol{\tau}^{[i]})}$$

with respect to the new policy parameters $\boldsymbol{\theta}$. This minimization is also called the *Moment-Projection* of the reward weighted trajectory distribution as it matches the moments of $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ with the moments of $p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})$. It forces $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ to have probability mass everywhere where $p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})$ has non-negligible probability mass. Consequently, if $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ is a Gaussian, the M-projection averages over all modes of the reward weighted trajectory distribution.

Alternatively, we can use the Information (I)-projection $\mathrm{argmin}_{\boldsymbol{\theta}}$ $\mathrm{KL}(p_{\boldsymbol{\theta}}(\boldsymbol{\tau})||p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau}))$ to update the policy, as introduced in the Variational Inference for Policy Search [48] algorithm.

This projection forces the new trajectory distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ to be zero everywhere where the reward weighted trajectory distribution is is zero. When using a Gaussian distribution for $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$, the I-projection will concentrate on a single mode of $p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})$ and lose information about the other modes contained in the samples. Unfortunately, it can not be determined in closed form for most distributions.

Variational Inference for Policy Search. In the Variational Inference for policy search approach [48], a parametric representation of the variational distribution $q_{\boldsymbol{\beta}}(\boldsymbol{\tau})$ is used instead of a sample-based approximation as used in the MC-EM approach. It is convenient to choose $q_{\boldsymbol{\beta}}(\tau)$ from the same family of distributions as $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$. Now, a sample-based approximation is used to replace the integral in the KL-divergence $\mathrm{KL}\left(q_{\boldsymbol{\beta}}(\boldsymbol{\tau})\ ||p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})\right)$ needed for the E-step, i.e.,

$$\begin{aligned}\boldsymbol{\beta} &\in \mathrm{argmin}_{\tilde{\boldsymbol{\beta}}}\mathrm{KL}(q_{\tilde{\boldsymbol{\beta}}}(\boldsymbol{\tau})||p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau})) \\ &\approx \mathrm{argmin}_{\tilde{\boldsymbol{\beta}}} \sum_{\boldsymbol{\tau}^{[i]}} q_{\tilde{\boldsymbol{\beta}}}(\boldsymbol{\tau}^{[i]}) \log \frac{q_{\tilde{\boldsymbol{\beta}}}(\boldsymbol{\tau}^{[i]})}{p(R|\boldsymbol{\tau}^{[i]})p_{\boldsymbol{\theta}'}(\boldsymbol{\tau}^{[i]})}\,. \end{aligned} \tag{2.60}$$

The minimization of this KL-divergence is equivalent to the *I-projection* of the reward-weighted trajectory distribution $p(R|\boldsymbol{\tau})p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$. In the variational approach, the M-step now trivially reduces to setting the new parameter vector $\boldsymbol{\theta}_{\text{new}}$ to $\boldsymbol{\theta}'$.

Hence, the MC-EM and the variational inference algorithm only differ in the employed projections of the reward-weighted trajectory distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|R)$. As the projections are in general different, they each converge to a different (local) maximum of the lower bound $\mathcal{L}_{\boldsymbol{\theta}}(q)$. The variational inference algorithm has been used in the episode-based formulation to learn an upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ for multiple contexts. If we use a Gaussian distribution for $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$, the I-projection concentrates on a single mode, as shown in Figure 2.4. Such behavior can be beneficial if all modes are almost equally good. However, the I-projection might also choose a sub-optimal mode (which has a lower reward). The M-projection averages over all modes, and, therefore, might also include large areas of low reward in the distribution. The behavior of both approaches for a simple multi-modal toy problem

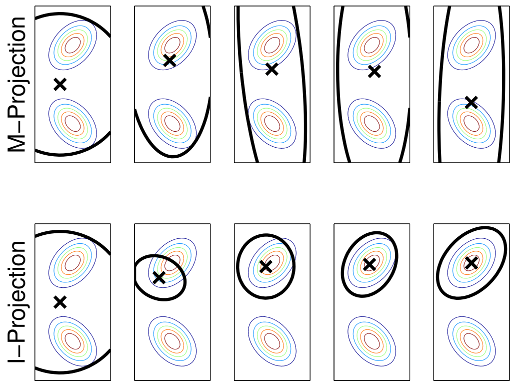


Fig. 2.4 Comparison of using the I-projection versus the M-projection for inference based policy search. While the M-projection averages over several modes of the reward function, the I-projection concentrates on a single mode, and, therefore, avoids including areas of low reward in the policy distribution.

is illustrated in Figure 2.4.

If the target distribution is uni-modal, both projections yield almost the same solutions. However, using the I-projection is computationally demanding, and, hence, the variational inference approach is generally not the method of choice. In addition, if we use a more complex distribution for modeling the policy, e.g., a mixture of Gaussians, the difference between the I- and the M-projection becomes less distinct.

2.4.3 Information-Theoretic Approaches

The main idea behind the information-theoretic approaches is to stay close to the 'data', i.e., the trajectory distribution after the policy update should not jump away from the trajectory distribution before the policy update. Information-theoretic approaches bound the distance between the old trajectory distribution $q(\boldsymbol{\tau})$ and the newly estimated trajectory distribution $p(\boldsymbol{\tau})$ at each update step. Such regularization of the policy update limits the information loss of the updates, and, hence, avoids that the new distribution $p(\boldsymbol{\tau})$ prematurely concentrates on local optima of the reward landscape. The first type of algorithms to implement this insight from information theory were the natural policy gradient algorithms [61], which have already been discussed in the policy gradient section. Natural policy gradient algorithms always require a user-specified learning rate, an issue which was alleviated by EM-based methods. However, EM-based methods have other problems concerning premature convergence and stability of the learning process as they typically do not stay close to the data. The information theoretic insight was again taken up in [57] with the Relative Entropy Policy Search (REPS) algorithm to combine the advantages of both types of algorithms. REPS uses the same information theoretic bound as the NAC algorithm but simultaneously updates its policy by weighted maximum likelihood estimates, which do not require a learning rate.

REPS formulates the policy search problem as an optimization problem, wherein the optimization is done directly in the space of distributions p over trajectories, state-actions pairs or parameters without considering a direct or indirect parametrization of p. As we will see, the REPS optimization problem allows for a closed-form solution for computing p. The used distance measure $\mathrm{KL}(p||q)$ forces p to be low everywhere where the old 'data' distribution q is also low. Intuitively, bounding $\mathrm{KL}(p||q)$ prevents p to 'move outside' the 'old data' distribution q as such behavior is potentially dangerous for the robot. The use of the opposite KL-divergence $\mathrm{KL}(q||p)$ would not exhibit this favorable property and also does not allow for a closed form solution for p.

2.4.3.1 Episode-based Relative Entropy Policy Search

We start our discussion with the episode-based formulation [17] of relative entropy policy search [57] as it is the simplest formulation. In the episode-based formulation, we need to learn an upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ for selecting the parameters of the lower-level policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)$ in order to maximize the average return $J_{\boldsymbol{\omega}}$ as defined in Equation (2.2). At the same time, we want to bound the Kullback-Leibler divergence between the newly estimated policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ and the old policy $q(\boldsymbol{\omega})$.

Resulting Optimization Program. To do so, we can solve the following constrained optimization problem

$$\begin{aligned}
&\max_{\pi} \int \pi(\boldsymbol{\theta}) R(\boldsymbol{\theta}) d\boldsymbol{\theta}, \\
\text{s.t.} \quad &\epsilon \geq \int \pi(\boldsymbol{\theta}) \log \frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta}, \\
&1 = \int \pi(\boldsymbol{\theta}) d\boldsymbol{\theta},
\end{aligned} \tag{2.61}$$

This constrained optimization problem can be solved efficiently by the method of Lagrangian multipliers. Please refer to Appendix C for more details. From the Lagrangian, we can also obtain a closed-form solution for the new policy

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta}) \exp\left(\frac{R(\boldsymbol{\theta})}{\eta}\right), \tag{2.62}$$

where η is the Lagrangian multiplier connected to the KL-bound constrained. It specifies a scaling factor for the reward and can be interpreted as the temperature of the soft-max distribution given in Equation (2.62).

The Dual Function. The parameter η is obtained by minimizing the dual-function $g(\eta)$ of the original optimization problem,

$$g(\eta) = \eta\epsilon + \eta \log \int q(\boldsymbol{\theta}) \exp\left(\frac{R(\boldsymbol{\theta}}{\eta}\right) d\boldsymbol{\theta}. \tag{2.63}$$

The derivation of the dual function is given in Appendix C. In practice, the integral in the dual function is approximated by samples, i.e.,

$$g(\eta) = \eta\epsilon + \eta \log \sum_i \frac{1}{N} \exp\left(\frac{R(\boldsymbol{\theta}^{[i]})}{\eta}\right) d\boldsymbol{\theta}. \tag{2.64}$$

Estimating the New Policy. The new policy $\pi(\boldsymbol{\theta})$ is also only known for samples $\boldsymbol{\theta}^{[i]}$ where we have evaluated the reward $R(\boldsymbol{\theta}^{[i]})$. Consequently, we need to fit a parametric distribution $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ to our samples. This parametric distribution is obtained by a weighted maximum likelihood estimate on the samples $\boldsymbol{\theta}^{[i]}$ with the weightings

$d^{[i]} = \exp\left(R(\boldsymbol{\theta}^{[i]})/\eta\right)$. Note that the distribution $q(\boldsymbol{\theta}^{[i]})$ can be dropped from the weighting as we have already sampled from q. Typically, the parametric policy is Gaussian, and, hence the new parameters are given by the weighted mean and covariance. For the resulting updates of the weighted maximum likelihood estimates please refer to Appendix B.

In theory, the expected return $R(\boldsymbol{\theta}^{[i]}) = \int p_{\boldsymbol{\theta}^{[i]}}(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau}$ is given as expectation over all possible roll-outs. However, to increase the sample efficiency in practice, the return $R(\boldsymbol{\theta}^{[i]})$ is typically approximated with a single sample. Similar to EM-based approaches, such strategy introduces a bias into the optimization problem, as the expectation is not performed inside the exp function, and, consequently, the resulting policy is risk-seeking. However, for moderately stochastic system no performance loss was observed.

Although it seems natural to define $q(\boldsymbol{x}, \boldsymbol{u})$ as the old policy $\pi(\boldsymbol{\omega}_{k-1})$, we can use the last K policies as $q(\boldsymbol{x}, \boldsymbol{u})$ to reuse samples from previous iterations.

2.4.3.2 Episode-Based Extension to Multiple Contexts.

In episode-based REPS, we can extend the upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ to select the policy parameters $\boldsymbol{\theta}$ based on the context $\boldsymbol{s}$. Our aim is to maximize the expected reward $\mathcal{R}_{\boldsymbol{s\theta}}$ while bounding the expected relative entropy between $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ and the old policy $q(u|x)$, i.e.,

$$\begin{aligned}&\max_{\pi} \int \mu(\boldsymbol{s}) \int \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})\mathcal{R}_{\boldsymbol{s\theta}} d\boldsymbol{\theta} d\boldsymbol{s}\\ &\text{s.t: } \epsilon \geq \int \mu(\boldsymbol{s})\mathrm{KL}\left(\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})||q(\boldsymbol{\theta}|\boldsymbol{s})\right) d\boldsymbol{s}, \quad \forall \boldsymbol{s} : 1 = \int \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) d\boldsymbol{\theta}.\end{aligned} \tag{2.65}$$

However, this formulation requires that we have access to many parameter vector samples $\boldsymbol{\theta}^{[i,j]}$ for a single context vector $\boldsymbol{s}^{[i]}$. In order to relax this assumption, contextual REPS optimizes for the joint probabilities $p(\boldsymbol{s}, \boldsymbol{\theta})$ and enforces that $p(\boldsymbol{s}) = \int p(\boldsymbol{s}, \boldsymbol{\theta})d\boldsymbol{\theta}$ still reproduces the correct context distribution $\mu(\boldsymbol{s})$ by using the constraints $\forall \boldsymbol{s} : p(\boldsymbol{s}) = \mu(\boldsymbol{s})$.

Matching Feature Averages. Yet, in continuous spaces, this formulation results in an infinite amount of constraints. Therefore, we need to resort to matching feature expectations instead of matching single probabilities, i.e., $\int p(\boldsymbol{s})\boldsymbol{\varphi}(\boldsymbol{s})d\boldsymbol{s} = \hat{\varphi\phi}$ where $\hat{\varphi\phi} = \int \mu(\boldsymbol{s})\boldsymbol{\varphi}(\boldsymbol{s})d\boldsymbol{s}$ is the observed average feature vector. For example, if φ contains all linear and quadratic terms of the context $\boldsymbol{s}$, we match the first and second order moments of both distribution, i.e., mean and variance.

Optimization Program for Contextual Policy Search. The resulting optimization program yields

$$\begin{aligned}
&\max_p \iint p(\boldsymbol{s},\boldsymbol{\theta})\mathcal{R}_{\boldsymbol{s\theta}} d\boldsymbol{\theta} d\boldsymbol{s} \\
&\text{s.t: } \epsilon \geq \iint p(\boldsymbol{s},\boldsymbol{\theta}) \log \frac{p(\boldsymbol{s},\boldsymbol{\theta})}{\mu(\boldsymbol{s})q(\boldsymbol{\theta}|\boldsymbol{s})} d\boldsymbol{\theta} d\boldsymbol{s} \\
&\qquad \hat{\boldsymbol{\phi}} = \iint p(\boldsymbol{s},\boldsymbol{\theta})\boldsymbol{\phi}(\boldsymbol{s}) d\boldsymbol{\theta} d\boldsymbol{s}, \quad 1 = \iint p(\boldsymbol{s},\boldsymbol{\theta}) d\boldsymbol{\theta} d\boldsymbol{s}.
\end{aligned} \tag{2.66}$$

Note that, by replacing $p(\boldsymbol{s})$ with $\mu(\boldsymbol{s})$, we end up with the original optimization problem from Equation (2.65). This optimization problem can be solved by the method of Lagrangian multipliers and yields a closed-form solution for $p(\boldsymbol{s},\boldsymbol{\theta})$ that is given by

$$p(\boldsymbol{s},\boldsymbol{\theta}) \propto q(\boldsymbol{\theta}|\boldsymbol{s})\mu(\boldsymbol{s}) \exp\left(\frac{\mathcal{R}_{\boldsymbol{s\theta}} - V(\boldsymbol{s})}{\eta}\right), \tag{2.67}$$

where $V(\boldsymbol{s}) = \boldsymbol{\phi}(\boldsymbol{s})^T\boldsymbol{v}$ is a context dependent baseline which is subtracted from the reward. The parameters η and $\boldsymbol{v}$ are Lagrangian multipliers which are obtained by minimizing the dual function $g(\eta,\boldsymbol{\theta})$ of the optimization problem [17]. The dual function is given in the Appendix C.

The function $V(\boldsymbol{s})$ also has an interesting interpretation, which can be obtained when looking at the optimality condition for $V(\boldsymbol{s}_i) = v_i$ for nominal context variables[3], $V_i = \eta \log \int q(\boldsymbol{\theta}|\boldsymbol{s}_i) \exp\left(\mathcal{R}_{\boldsymbol{s}_i\boldsymbol{\theta}}/\eta\right) d\boldsymbol{\theta}$, and, hence, $V(\boldsymbol{s}_i)$ is given by a soft-max operator over the expected rewards in context $\boldsymbol{s}_i$. Consequently, $V(\boldsymbol{s}_i)$ can be interpreted as value function [57].

[3] In this case, we do not have to use features.

```
**Algorithm 12** Episode-Based REPS Updates for Multiple Contexts
  **Input:** KL-bounding $\epsilon$
        data-set $\mathcal{D}_{ep} = \left\{ \boldsymbol{s}^{[i]}, \boldsymbol{\theta}^{[i]}, R^{[i]} \right\}_{i=1\ldots N}$
  **Optimize dual-function** $[\eta, \boldsymbol{v}] = \text{argmin}_{\eta', \boldsymbol{v}'} g(\eta', \boldsymbol{v}'), \quad \text{s.t. } \eta > 0$
      $g(\eta, \boldsymbol{v}) = \eta\epsilon + \boldsymbol{v}^T \hat{\boldsymbol{\varphi}} + \eta \log \left( \sum_{i=1}^{N} \frac{1}{N} \exp \left( \frac{R^{[i]} - \boldsymbol{v}^T \boldsymbol{\varphi}\left(\boldsymbol{s}^{[i]}\right)}{\eta} \right) \right)$
  **Obtain parametric policy** $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ by weighted ML estimate
      $\boldsymbol{\omega}_{\text{new}} = \text{argmax}_{\boldsymbol{\omega}} \sum_{i=1}^{N} \exp \left( \frac{R^{[i]} - \boldsymbol{v}^T \boldsymbol{\varphi}\left(\boldsymbol{s}^{[i]}\right)}{\eta} \right) \log \pi_{\boldsymbol{\omega}}\left(\boldsymbol{\theta}^{[i]}|\boldsymbol{s}^{[i]}\right)$
```

The dual function and the new policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$ is again computed based on samples. Subsequently, a new parametric distribution is obtained by performing a weighted maximum likelihood (ML) estimate. Typically, a linear Gaussian policy is used to represent the upper-level policy. The weighted ML updates for this policy are given in Appendix B. The episode-based REPS algorithm for generalizing the upper-level policy to multiple contexts is given in Algorithm 12.

2.4.3.3 Learning Multiple Solutions with REPS

Using maximum likelihood estimates for the parameter updates is also beneficial for learning multiple solutions of a motor task as we can represent these multiple solutions as mixture model [17]. REPS can be extended to learning multiple solutions by reformulating the problem as a latent variable estimation problem. The upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ is now extended with another layer of hierarchy that consists of a gating-policy $\pi(o|\boldsymbol{s})$ which selects the option o to execute given the current context $\boldsymbol{s}$. Subsequently, the option policy $\pi(\boldsymbol{\theta}|\boldsymbol{s}, o)$ selects the parameter vector $\boldsymbol{\theta}$ of the lower level policy which controls the robot. The upper-level policy can now be written as mixture model, i.e.,

$$\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) = \sum_{o} \pi(o|\boldsymbol{s}) \pi(\boldsymbol{\theta}|\boldsymbol{s}, o). \tag{2.68}$$

With such a hierarchical approach, we can represent multiple solutions for the same motor task, such as, fore-hand and back-hand strokes in robot table tennis [38]. The gating policy allows inferring which options are feasible for context, allowing us to construct complex policies out of simpler 'option policies'.

Hierarchical Policy Search as a Latent Variable Estimation Problem. For efficient data-usage, we have to allow parameter vectors $\boldsymbol{\theta}$ from other options o' to be used to update the option policy $\pi(\boldsymbol{\theta}|\boldsymbol{s}, o)$ of option o. To achieve this goal, the options are treated as latent variables. Hence, only $q(\boldsymbol{s}, \boldsymbol{\theta})$ can be accessed and not $q(\boldsymbol{s}, \boldsymbol{\theta}, o)$. The bound on the KL can still be written in terms of the joint distribution $p(\boldsymbol{s}, \boldsymbol{\theta}, o) = p(\boldsymbol{s})\pi(\boldsymbol{s}|o)\pi(\boldsymbol{\theta}|\boldsymbol{s}, o)$ as

$$\epsilon \geq \sum_o \iint p(\boldsymbol{s}, \boldsymbol{\theta}, o) \log \left(\frac{p(\boldsymbol{s}, \boldsymbol{\theta}, o)}{q(\boldsymbol{s}, \boldsymbol{\theta}) p(o|\boldsymbol{s}, \boldsymbol{\theta})} \right) d\boldsymbol{s} d\boldsymbol{\theta}, \tag{2.69}$$

where $p(o|\boldsymbol{s}, \boldsymbol{\theta})$ is obtained by Bayes theorem. Furthermore, options should not overlap as we want to learn distinct solutions for the motor task. As a measure for the overlap of the options, the expected entropy of $p(o|\boldsymbol{s}, \boldsymbol{\theta})$ is used, i.e.,

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{\theta}} \left[H(p(o|\boldsymbol{s}, \boldsymbol{\theta}) \right] = -\sum_o \iint p(\boldsymbol{s}, \boldsymbol{\theta}, o) \log p(o|\boldsymbol{s}, \boldsymbol{\theta}) d\boldsymbol{s} d\boldsymbol{\theta}. \tag{2.70}$$

The overlap of the options should decrease by a certain percentage in each policy update step. Hence, the following constraint is introduced

$$\kappa \geq \mathbb{E}_{\boldsymbol{s},\boldsymbol{\theta}} \left[H(p(o|\boldsymbol{s}, \boldsymbol{\theta}) \right]. \tag{2.71}$$

The upper bound κ is usually set as a percentage of the currently measured overlap $\hat{H}_q$, i.e., $\kappa = \hat{H}_q \tilde{\kappa}$, where $1 - \tilde{\kappa}$ denotes the desired decrease of the overlap. Figure 2.5 illustrates the resulting policy updates with and without bounding the overlap on a simple multi-modal reward function. Without bounding the overlap of the options, both options concentrate on both modes of the reward function. As a consequence, the quality of both options is rather poor. By introducing the overlap constraint, both options separate early in the optimization process, and, thus, concentrate on the individual modes.

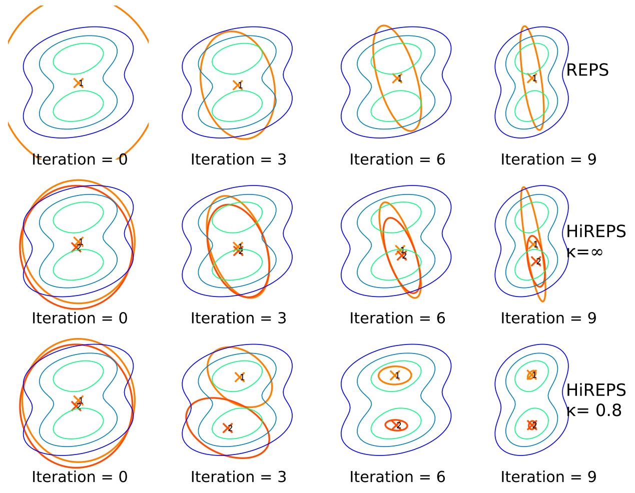


Fig. 2.5 Comparison REPS and HiREPS with and without bounding the overlap of the options on a simple bimodal reward function. The standard REPS approach only uses one option which averages over both modes. The HiREPS approach can use multiple options (two are shown). However, if we do not bound the overlap ($\kappa = \infty$), the options do not separate and concentrate on both modes. Only if we use the overlap constraint, we get a clear separation of the policies.

Lower Bound for the Optimization Problem with Latent Variables. Putting together the new constraints, HiREPS is defined as the following optimization problem:

$$\max_p \sum_o \iint p(\boldsymbol{s}, \boldsymbol{\theta}, o)\mathcal{R}_{\boldsymbol{s\theta}} d\boldsymbol{s} d\boldsymbol{\theta},$$

$$\text{s. t.} \quad \epsilon \geq \sum_o \iint p(\boldsymbol{s}, \boldsymbol{\theta}, o) \log \left(\frac{p(\boldsymbol{s}, \boldsymbol{\theta}, o)}{q(\boldsymbol{s}, \boldsymbol{\theta}) p(o|\boldsymbol{s}, \boldsymbol{\theta})} \right) d\boldsymbol{s} d\boldsymbol{\theta}, \tag{2.72}$$

$$\hat{\boldsymbol{\phi}} = \iint p(\boldsymbol{s}, \boldsymbol{\theta}) \boldsymbol{\phi}(\boldsymbol{s}) d\boldsymbol{\theta} d\boldsymbol{s}, \quad 1 = \iint p(\boldsymbol{s}, \boldsymbol{\theta}) d\boldsymbol{\theta} d\boldsymbol{s},$$

$$\kappa \hat{H}_q \geq \mathbb{E}_{\boldsymbol{s}, \boldsymbol{\theta}} \left[H(p(o|\boldsymbol{s}, \boldsymbol{\theta})) \right],$$

$$1 = \sum_o \iint p(\boldsymbol{s}, \boldsymbol{\theta}, o) d\boldsymbol{s} d\boldsymbol{\theta}. \tag{2.73}$$

Unfortunately, this optimization problem can not be solved in closed form as it contains the conditional distribution $p(o|\boldsymbol{s}, \boldsymbol{\theta})$ inside the logarithm. However, a lower bound of this optimization problem can be obtained using an EM-like update procedure [17]. In the E-step, we estimate

$$\tilde{p}(o|\boldsymbol{s},\boldsymbol{\theta}) = \frac{p(\boldsymbol{s},\boldsymbol{\theta},o)}{\sum_o p(\boldsymbol{s},\boldsymbol{\theta},o)}.$$

In the M-step, we use $\tilde{p}(o|\boldsymbol{s},\boldsymbol{\theta})$ for $p(o|\boldsymbol{s},\boldsymbol{\theta})$ in the optimization problem, and, therefore, neglect the relationship between $p(o|\boldsymbol{s},\boldsymbol{\theta})$ and the joint distribution $p(\boldsymbol{s},\boldsymbol{\theta},o)$. Similar to EM, it can be shown that this iterative optimization procedure maximizes a lower bound of the original optimization problem that is tight after each E-step [17]. The resulting joint distribution has the following solution

$$p(\boldsymbol{s},\boldsymbol{\theta},o) \propto q(\boldsymbol{s},\boldsymbol{\theta})\tilde{p}(o|\boldsymbol{s},\boldsymbol{\theta})^{1+\kappa/\eta} \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\theta}} - \varphi(\boldsymbol{s})^T\boldsymbol{v}}{\eta}\right), \qquad (2.74)$$

where κ denotes the Lagrangian multiplier from bounding the overlap of the options, see Equation (2.71).

The dual function $g(\eta,\boldsymbol{v})$ that is needed to obtain the parameters η and $\boldsymbol{v}$, is given in the appendix C. As described in the previous section, the dual-function is approximated with samples and the probabilities $p(\boldsymbol{s},\boldsymbol{\theta},o)$ are only known for the given set of sample. Hence, we need to fit parametric models to the gating policy as well as to the option policies. Simple linear Gaussian models were used to represent the option policies $\pi(\boldsymbol{\theta}|o,\boldsymbol{s})$ and the gating policy was also given by a Gaussian gating. The episode-based HiREPS algorithm is summarized in Algorithm 13.

The advantage of using weighted maximum likelihood policy updates for determining hierarchical policies has yet still to be explored for more complex hierarchies. Exploiting structures such as hierarchies might well be the missing key to scale robot learning to more complex real world environments.

2.4.3.4 Step-based REPS for Infinite Horizon Problems

The original REPS formulation [57] is step-based and uses an infinite horizon formulation. The step-based algorithm uses the KL divergence $\mathrm{KL}(p(\boldsymbol{x},\boldsymbol{u})||q(\boldsymbol{x},\boldsymbol{u}))$ on the resulting distribution over the state-action

Algorithm 13 Episode-Based HiREPS for Multiple Contexts

```
**Input:** KL-bounding $\epsilon$, overlap-bounding $\kappa$
    data-set $\mathcal{D}_{\text{ep}} = \left\{\boldsymbol{s}^{[i]}, \boldsymbol{\theta}^{[i]}, R^{[i]}\right\}_{j=1\dots N}$
    old gating: $q(o|\boldsymbol{s})$, old option-policies $q(\boldsymbol{\theta}|\boldsymbol{s}, o)$
**Compute** $p(o|\boldsymbol{s}, \boldsymbol{\theta})$ for all options and samples $i$
    $\tilde{p}\,(o|i) = \frac{q(\boldsymbol{\theta}^{[i]}|\boldsymbol{s}^{[i]}, o)q(o|\boldsymbol{s}^{[i]})}{\sum_{o'} q(\boldsymbol{\theta}^{[i]}|\boldsymbol{s}^{[i]}, o')q(o'|\boldsymbol{s}^{[i]})}$
**Optimize dual-function**, see Equation (4.20)
    $[\eta, \boldsymbol{v}] = \text{argmin}_{\eta', \boldsymbol{v}'} g(\eta', \boldsymbol{v}'), \quad \text{s.t. } \eta > 0$
**Obtain option policies** $\pi^O_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}, o)$ for all options $o$
    $\boldsymbol{\omega}^o_{\text{new}} = \text{argmax}_{\boldsymbol{\omega}} \sum_{i=1}^{N} d_o^{[i]} \log \pi^O_{\boldsymbol{\omega}}\left(\boldsymbol{\theta}^{[i]}|\boldsymbol{s}^{[i]}, o\right)$
**Obtain gating policy** $\pi^G_{\boldsymbol{\omega}}(o|\boldsymbol{s})$ by weighted ML estimate
    $\boldsymbol{\omega}^G_{\text{new}} = \text{argmax}_{\boldsymbol{\omega}} \sum_{i=1}^{N} \sum_{o} d_o^{[i]} \log \pi^G_{\boldsymbol{\omega}}\left(o|\boldsymbol{s}^{[i]}\right)$
with $d_o^{[i]} = \tilde{p}\,(o|i)^{1+\xi/\eta} \exp\left(\frac{R^{[i]} - \boldsymbol{v}^T\boldsymbol{\varphi}\left(\boldsymbol{s}^{[i]}\right)}{\eta}\right)$
```

pairs $p(\boldsymbol{x}, \boldsymbol{u})$, as a similarity measure of the new trajectory distribution $p(\boldsymbol{\tau})$ and the old trajectory distribution $q(\boldsymbol{\tau})$.

In the infinite horizon formulation, REPS maximizes the average reward per time step, given as

$$J_{\pi,\mu^\pi} = \mathbb{E}[r(\boldsymbol{x}, \boldsymbol{u})] = \iint \mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})r(\boldsymbol{x}, \boldsymbol{u})d\boldsymbol{x}d\boldsymbol{u}. \tag{2.75}$$

The distribution $\mu^\pi(\boldsymbol{x})$ denotes the stationary state distribution of the MDP with policy π.

Stationary State Distributions. The stationary state distribution $\mu^\pi(\boldsymbol{x})$ represents the probability of visiting state $\boldsymbol{x}$ when following policy π. It can not be chosen freely but has to comply with the given state dynamics and the policy. Therefore, it has to fulfill the following constraint

$$\forall \boldsymbol{x}' : \mu^\pi(\boldsymbol{x}') = \iint_{\boldsymbol{x},\boldsymbol{u}} \mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})d\boldsymbol{x}d\boldsymbol{u}. \tag{2.76}$$

As these constraints are not feasible for continuous state spaces, we can again require that the distributions only match on their expected state-features $\boldsymbol{\varphi}(\boldsymbol{x})$, i.e. the expected feature from the distribution $\mu^\pi(\boldsymbol{x}')$ need to match the expected features of the distribution

$$\tilde{\mu}^\pi(\boldsymbol{x}') = \iint \mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})d\boldsymbol{x}\boldsymbol{u},$$

which corresponds to $\mu^\pi(\boldsymbol{x}')$ after applying the policy and the system dynamics. Such a constraint can be formalized as

$$\int \mu^\pi(\boldsymbol{x}')\boldsymbol{\varphi}(\boldsymbol{x}')d\boldsymbol{x}' = \iiint \mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})\boldsymbol{\varphi}(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{u}d\boldsymbol{x}'. \tag{2.77}$$

Closeness-to-the-Data Constraint. To ensure the closeness to the old distribution, we bound the relative entropy between the old state-action distribution $q(x,u)$ and the new state action distribution $\mu^\pi(\boldsymbol{x})\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$, i.e.,

$$\begin{aligned}\epsilon &\geq \mathrm{KL}(\mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})||q(\boldsymbol{x},\boldsymbol{u})) \\ &= \iint \mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})\log\frac{\mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})}{q(\boldsymbol{x},\boldsymbol{u})}d\boldsymbol{x}d\boldsymbol{u}. \end{aligned} \tag{2.78}$$

This bound again limits the loss of information and ensures a smooth learning progress.

Resulting Optimization Program. The resulting optimization program can be formalized as follows:

$$\max_{\pi,\mu^\pi} \iint \mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})r(\boldsymbol{x},\boldsymbol{u})d\boldsymbol{x}d\boldsymbol{u}, \tag{2.79}$$

$$\text{s. t.}: \quad \epsilon \geq \sum_{x,u} \mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x}) \log \frac{\mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})}{q(\boldsymbol{x},\boldsymbol{u})} d\boldsymbol{x} d\boldsymbol{u}\,,$$

$$\int \mu^\pi(\boldsymbol{x}')\boldsymbol{\varphi}(\boldsymbol{x}')d\boldsymbol{x}' = \iiint \mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})\boldsymbol{\varphi}(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{u}d\boldsymbol{x}',$$

$$1 = \iint \mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})d\boldsymbol{x}d\boldsymbol{u},$$

where the last constraint ensures that $\mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})$ is a normalized probability distribution. This constrained optimization problem can again be solved efficiently by the method of Lagrangian multipliers. Please refer to Appendix C for more details. From the Lagrangian, we can also obtain a closed-form solution for the state-action distribution $\mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})$ which is given as

$$\mu^\pi(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x}) \propto q(\boldsymbol{x},\boldsymbol{u}) \exp\left(\frac{r(\boldsymbol{x},\boldsymbol{u}) + \mathbb{E}_{\boldsymbol{x}'}[V(\boldsymbol{x}')] - V(\boldsymbol{x})}{\eta}\right) \quad (2.80)$$

for the joint distribution, and as

$$\pi(\boldsymbol{u}|\boldsymbol{x}) \propto q(\boldsymbol{u}|\boldsymbol{x}) \exp\left(\frac{r(\boldsymbol{x},\boldsymbol{u}) + \mathbb{E}_{\boldsymbol{x}'}[V(\boldsymbol{x}')]}{\eta}\right) \quad (2.81)$$

for the policy $\pi(\boldsymbol{u}|\boldsymbol{x})$. The parameter η denotes the Lagrangian multiplier for the relative entropy bound and the the function $V(\boldsymbol{x}) = \boldsymbol{\varphi}^T(\boldsymbol{x})\boldsymbol{v}$ includes the Lagrangian multipliers $\boldsymbol{v}$ for the stationary distribution constraint from Equation (2.77).

The Lagrangian parameters can be efficiently obtained by minimizing the dual function $g(\eta, \boldsymbol{v})$ of the optimization problem. Intuitively, $V(\boldsymbol{x})$ can be seen as a value-function. As we can see, the expected value $\mathbb{E}_{p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})}[V(\boldsymbol{x}')]$ of the next state is added to the reward while the current value $V(\boldsymbol{x})$ is subtracted. With this interpretation, we can also interpret the term $\delta_V(\boldsymbol{x},\boldsymbol{u}) = r(\boldsymbol{x},\boldsymbol{u}) + \mathbb{E}_{p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})}[V(\boldsymbol{x}')] - V(\boldsymbol{x})$ as advantage function of state-action pair $(\boldsymbol{x},\boldsymbol{u})$. Hence, we now use the advantage function to determine the exponential weighting of the state-action pairs. The function $V(\boldsymbol{x})$ is highly connected to the policy gradient baseline. However, the REPS baseline directly emerged out of the derivation of the algorithm while it has to be added afterwards for the policy gradient algorithms to decrease the variance of the gradient estimate.

The Dual Function. The dual function $g(\eta, \boldsymbol{V})$ of the step-based REPS optimization problem is given in *log-sum-exp* form

$$g(\eta, \boldsymbol{V}) = \eta\epsilon + \eta \log \iint q(\boldsymbol{x}, \boldsymbol{u}) \exp\left(\frac{\delta_V(\boldsymbol{x}, \boldsymbol{u})}{\eta}\right) d\boldsymbol{x} d\boldsymbol{u}. \tag{2.82}$$

Due to its *log-sum-exp* form, the dual-function is convex in $\boldsymbol{V}$. Furthermore, we can approximate the expectation over $q(x, u)$ with a sum over samples $(x^{[i]}, u^{[i]})$ from the distribution $q(x, u)$, i.e.,

$$g(\eta, \boldsymbol{V}) \approx \eta\epsilon + \eta \log \frac{1}{N} \sum_{\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}} \exp\left(\frac{\delta_V(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]})}{\eta}\right). \tag{2.83}$$

Consequently, we do not need to know the distribution $q(\boldsymbol{x}, \boldsymbol{u})$ as a function, but only need to be able to sample from it. The parameters η and $\boldsymbol{v}$ are obtained by minimizing the dual function. As η results from an inequality constraint, η needs to be larger than zero [14]. Solving the dual optimization problem is therefore given by the following program $[\eta, \boldsymbol{v}] = \mathrm{argmin}_{\eta', \boldsymbol{v}'} g(\eta', \boldsymbol{v}'), \quad \text{s.t.: } \eta' > 0.$

Estimating the New Policy. To deal with continuous actions, we need to use a parametric policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$. Similar as to the episode-based algorithm, we can compute the probabilities $\mu^\pi(\boldsymbol{x}^{[i]})\pi(\boldsymbol{u}^{[i]}|\boldsymbol{x}^{[i]})$ only for the given set of samples and subsequently fit a parametric distribution $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ to these samples. Fitting the policy corresponds to a weighted maximum likelihood estimate where the weighting is given by $d^{[i]} = \exp\big(\delta_V(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]})/\eta\big)$.

From this result, we can observe the close relationship with step-based EM-based policy search algorithms. For general reward functions, EM-based algorithms use an exponential transformation of the expected future return, i.e., $d^{[i]} = \exp\big(\beta Q^\pi(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]})\big)$, where the inverse temperature β has to be chosen by the user. In contrast, REPS always returns the optimized temperature η of the exponential weighting which exactly corresponds to the desired KL-bound. Note that the scaling η will be different for each policy update depending on the distribution of the current reward samples. Furthermore, REPS uses the value-function as a baseline to account for different achievable values in different states.

Representing the Transition Dynamics. The step-based REPS formulation requires the estimation of $\mathbb{E}_{p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})}[V(\boldsymbol{x}')]$, which would require the knowledge of the model $p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})$. However, in the current implementation [57], a single sample $\boldsymbol{x}'^{[i]}$ from the observed transition is used to approximate $\mathbb{E}_{p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})}[V(\boldsymbol{x}')]$, and hence, no model needs to be known. Such approximation causes a bias as the expectation is not done inside the exponential function which is used for computing $\pi(\boldsymbol{u}|\boldsymbol{x})$, and, hence, the policy does not optimize the average reward any more. However, in our experience, this bias has only a minor effect on the quality of the learned solution if the noise in the system is not too high. We summarize the REPS algorithm for the infinite horizon formulation in Algorithm 14. The algorithm computes the expected feature change for each state action pair. However, for continuous states and actions, each state action pair is typically only visited once, and, hence the expectation is approximated using a single sample estimate. Instead of using q as the state-action distribution of the old policy, we can also reuse samples by assuming that $q(\boldsymbol{x},\boldsymbol{u})$ is the state-action distribution of the last K policies.

2.4.4 Miscellaneous Important Methods

In this section, we will cover two types of algorithms, stochastic optimization and policy improvements with path integrals, which lead to promising results in robotics. For the stochastic optimization algorithm, we will discuss the Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) algorithm while for the path integral approach, we will discuss the Policy Improvements with Path Integral (PI^2) algorithm, both of which are well-known in the field of robot learning.

2.4.4.1 Stochastic Optimization

Stochastic optimizers are black-bock optimizers, and, hence, can be straightforwardly applied for policy search in the episode-based formulation. As it is typically the case with episode-based algorithms, they model a upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ to create samples in the parameter-space, which are subsequently evaluated on the real system.

Algorithm 14 REPS for infinite horizon problems

```
Input: KL-bounding $\epsilon$
        data-set $\mathcal{D} = \left\{ \boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}, r^{[i]}, \boldsymbol{x}'^{[i]} \right\}_{i=1\ldots N}$
for $i = 1 \ldots N$ do
    State-action visits: $n(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}) = \sum_j I_{ij}$
    Summed reward: $\tilde{r}(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}) = \sum_j I_{ij} r^{[j]}$
    Summed features: $\delta\tilde{\boldsymbol{\varphi}}\left(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}\right) = \sum_j I_{ij}\left(\boldsymbol{\varphi}(\boldsymbol{x}'^{[j]}) - \boldsymbol{\varphi}(\boldsymbol{x}^{[j]})\right)$
end for  ($I_{ij}$ is 1 if $\boldsymbol{x}^{[i]} = \boldsymbol{x}^{[j]}$ and $\boldsymbol{u}^{[i]} = \boldsymbol{u}^{[j]}$, 0 elsewhere)
Compute sample bellman error
    $$\delta_{\boldsymbol{v}}^{[i]} = \frac{\tilde{r}(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}) + \boldsymbol{v}^T \delta\tilde{\boldsymbol{\varphi}}(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]})}{n(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]})}$$
Optimize dual-function  $[\eta, \boldsymbol{v}] = \mathrm{argmin}_{\eta', \boldsymbol{v}'} g(\eta', \boldsymbol{v}')$,   s.t. $\eta > 0$
    $$g(\eta, \boldsymbol{v}) = \eta\epsilon + \eta \log\left(\frac{1}{N}\sum_{i=1}^{N} \exp\left(\frac{\delta_{\boldsymbol{v}}^{[i]}}{\eta}\right)\right)$$
Obtain parametric policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ by weighted ML estimate
    $$\boldsymbol{\theta}_{k+1} = \mathrm{argmax}_{\theta} \sum_{i=1}^{N} \exp\left(\frac{\delta_{\boldsymbol{v}}^{[i]}}{\eta}\right) \log \pi_{\boldsymbol{\theta}}\left(\boldsymbol{u}^{[i]} \middle| \boldsymbol{x}^{[i]}\right)$$
```

The Covariance Matrix Adaptation - Evolutionary Strategy. The Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) is considered as the state of the art in stochastic optimization [28]. CMA-ES was applied to policy search in robotics [30] and yielded promising results on standard benchmark tasks such as a cart-pole balancing task with two poles. The procedure of CMA-ES is similar to many episode-based policy search methods such as episode-based EM or episode-based REPS. CMA-ES maintains a Gaussian distribution $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ over the parameter vector $\boldsymbol{\theta}$, and uses the data-set $\mathcal{D}_{\mathrm{ep}}$ for the policy updates. Similar to the EM-based approaches, CMA-ES also uses a weight $d^{[i]}$ for each sample. However, for estimating the weight $d^{[i]}$ and updating the distribution $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ CMA-ES uses heuristics, which often work well in practice but are not founded on a theoretical basis. For estimating the weight $d^{[i]}$, CMA-ES first sorts the samples $\boldsymbol{\theta}^{[i]}$ according to their return $R^{[i]}$, and, subsequently, computes the weight of the best l samples by $d^{[i]} = \log(l+1) - \log(i)$. All other samples are neglected, i.e., get zero weight. Similar to weighted ML updates, the new mean $\boldsymbol{\mu}_k$ of policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ is computed by the weighted average of the data points. However, the update of the covariance matrix Σ is based on a combination of the weighted sample-covariance and information about the 'evolution path' $\{\boldsymbol{\mu}_j\}_{j=0\ldots k}$.

The advantage of such an approach is that the covariance matrix update is depends only on the current set of samples and, hence, requires only a few samples $\boldsymbol{\theta}^{[i]}$. The number of samples N to evaluate for CMA-ES is typically fixed to $\max(4 + 3\log D, 5)$, where D is the dimensionality of $\boldsymbol{\theta}$ and the number of samples l used for the weighting is typically set to $N/2$. While CMA-ES is a black-box optimizer and, therefore, simple to use, it also has severe disadvantages. It cannot be used for generalizing the upper-level policy to multiple contexts. Furthermore, several roll-outs have to be evaluated if the evaluation $R^{[i]}$ is noisy. The minimum number of required roll-outs for a given parameter $\boldsymbol{\theta}^{[i]}$ can be computed by Hoeffding and Bernstein races [29], which can slightly alleviate this problem. For a more detailed discussion about the CMA-ES updates we refer to [28].

2.4.4.2 Policy Improvement by Path Integrals

The Policy Improvements by Path Integrals (PI^2) algorithm [81] is based on the path integral approach to optimal control. The path integral approach is designed for obtaining optimal control laws for nonlinear continuous time systems of the form

$$\dot{\boldsymbol{x}}_t = \boldsymbol{f}(\boldsymbol{x}_t) + \boldsymbol{G}(\boldsymbol{x}_t)(\boldsymbol{u}_t + \boldsymbol{C}_u\boldsymbol{\epsilon}_t) = \boldsymbol{f}_t + \boldsymbol{G}_t(\boldsymbol{u}_t + \boldsymbol{\epsilon}_t), \tag{2.84}$$

where $\boldsymbol{f}_t$ denotes a drift term, $\boldsymbol{G}_t$ the control matrix of the system, $\boldsymbol{\epsilon}_t$ is zero mean Brownian motion and $\boldsymbol{C}_u$ is the diffusion coefficient. Note that these assumptions do not limit the generality of the approach as all physical systems linearly depend on the control action $\boldsymbol{u}_t$. Furthermore, the path integral theory assumes squared control costs of the form

$-\boldsymbol{u}_t\boldsymbol{R}\boldsymbol{u}_t$. The state-dependent part of the reward $r_t(\boldsymbol{x}_t)$ can be an arbitrary function. As path integrals are based on stochastic optimal control theory [83], we will now briefly review the relevant concepts. The stochastic optimal control (SOC) problem is now defined as finding the controls $\boldsymbol{u}_{1:T}$ which maximize the expected return

$$J(\boldsymbol{x}_1, \boldsymbol{u}_{1:T}) = r_T(\boldsymbol{x}_T) + \int_{t=0}^{T} r_t(\boldsymbol{x}_t, \boldsymbol{u}_t)dt. \tag{2.85}$$

The discrete time-formulation of the system for a fixed time step dt is given as

$$\boldsymbol{x}_{t+\mathrm{dt}} = \boldsymbol{x}_t + \boldsymbol{f}_t\mathrm{dt} + \boldsymbol{G}_t\left(\boldsymbol{u}_t\mathrm{dt} + \boldsymbol{C}_u\boldsymbol{\epsilon}_t\sqrt{\mathrm{dt}}\right), \tag{2.86}$$

where the term $\sqrt{\mathrm{dt}}$ appears because the variance of Brownian motion grows linearly with time, and, thus, the standard deviation grows with $\sqrt{\mathrm{dt}}$. The probability of the next state given the action and the previous state can also be written down as Gaussian distribution

$$p(\boldsymbol{x}_{t+\mathrm{dt}}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}\left(\boldsymbol{x}_t + \boldsymbol{f}_t\mathrm{dt} + \boldsymbol{G}_t\boldsymbol{u}_t\mathrm{dt}, \boldsymbol{G}_t\boldsymbol{\Sigma}_u\boldsymbol{G}_t^T\mathrm{dt}\right) \tag{2.87}$$

with $\boldsymbol{\Sigma}_u = \boldsymbol{C}_u\boldsymbol{C}_u^T$. The expected return for the discrete time formulation is given as

$$J(\boldsymbol{x}_1, \boldsymbol{u}_{1:T}) = r_T(\boldsymbol{x}_T) + \mathrm{dt}\left(\sum_{t=0}^{T} r_t(\boldsymbol{x}_t) - \boldsymbol{u}_t\boldsymbol{R}_t\boldsymbol{u}_t\right). \tag{2.88}$$

The discrete time formulations are needed for the derivation of the Hamilton-Jacobi Bellman (HJB) equation.

Hamilton-Jacobi Bellman Equation. The HJB Equation states the optimality conditions for a value function for continuous time systems as the one given in Equation (2.84). We start the derivation of the continuous time Bellman equation by with the discrete time system, and, after stating the optimality conditions for the discrete time system we will take the limit of dt $\to 0$ to get the continuous time formulation. The Bellman Equation for the discrete-time problem is given by

$$V(\boldsymbol{x}, t) = r_t(\boldsymbol{x}_t)\mathrm{dt} + \max_{\boldsymbol{u}}\left(-\boldsymbol{u}_t\boldsymbol{R}_t\boldsymbol{u}_t\mathrm{dt} + \mathbb{E}_{x_{t+\mathrm{dt}}}[V(\boldsymbol{x}_{t+\mathrm{dt}}, t + \mathrm{dt})]\right),$$

where the expectation is done with respect to system dynamics $p(\boldsymbol{x}_{t+\mathrm{dt}}|\boldsymbol{x}_t, \boldsymbol{u}_t)$. The HJB equation is now derived by using a second order Taylor approximation of the value function for time step $t + \mathrm{dt}$,

$$V(\boldsymbol{x} + \delta\boldsymbol{x}, t + \mathrm{dt}) \approx V(\boldsymbol{x}, t + \mathrm{dt}) + \delta\boldsymbol{x}^T\boldsymbol{v}_{\boldsymbol{x}} + \frac{1}{2}\delta\boldsymbol{x}^T\boldsymbol{V}_{\boldsymbol{x}\boldsymbol{x}}\delta\boldsymbol{x}, \tag{2.89}$$

with $\boldsymbol{v}_{\boldsymbol{x}} = \partial V(\boldsymbol{x}, t + \mathrm{dt})/\partial\boldsymbol{x}$ and $\boldsymbol{V}_{\boldsymbol{x}\boldsymbol{x}} = \partial V(\boldsymbol{x}, t + \mathrm{dt})/\partial^2\boldsymbol{x}$. As $V(\boldsymbol{x}, t+\mathrm{dt})$ is now represented as quadratic function and $p(\boldsymbol{x}_{t+\mathrm{dt}}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ is Gaussian, we can solve the expectation over the next state analytically

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}_{t+\mathrm{dt}}}[V(\boldsymbol{x}_{t+\mathrm{dt}}, t + \mathrm{dt})]) = V(\boldsymbol{x}, t + \mathrm{dt}) + (\boldsymbol{f}_t + \boldsymbol{G}_t\boldsymbol{u}_t)^T\boldsymbol{v}_{\boldsymbol{x}}\mathrm{dt}\\ + \frac{1}{2}\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{x}}\boldsymbol{V}_{\boldsymbol{x}\boldsymbol{x}}\mathrm{dt}) + O(\mathrm{dt}^2)\end{aligned} \tag{2.90}$$

with $\boldsymbol{\Sigma}_{\boldsymbol{x}} = \boldsymbol{G}_t\boldsymbol{\Sigma}_u\boldsymbol{G}_t^T$. Note that, the second order derivative $\boldsymbol{V}_{\boldsymbol{x}\boldsymbol{x}}$ also appears in the first order approximation of $\mathbb{E}_{\boldsymbol{x}_{t+\mathrm{dt}}}[V_{t+\mathrm{dt}}(\boldsymbol{x}_{t+\mathrm{dt}})])$. This term is a recurrent theme in stochastic calculus and directly relates to the Ito lemma [83]. Setting Equation (2.90) back into Equation (2.4.4.2), we get the following relationship

$$\begin{aligned}V(\boldsymbol{x}, t) = V(\boldsymbol{x}, t + \mathrm{dt}) + r_t(\boldsymbol{x}_t)\mathrm{dt}\\ + \max_{\boldsymbol{u}}\Big(- \boldsymbol{u}_t\boldsymbol{R}_t\boldsymbol{u}_t\mathrm{dt} + (\boldsymbol{f}_t + \boldsymbol{G}_t\boldsymbol{u}_t)^T\boldsymbol{v}_{\boldsymbol{x}}\mathrm{dt} + \frac{1}{2}\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{x}}\boldsymbol{V}_{\boldsymbol{x}\boldsymbol{x}}\mathrm{dt})\Big).\end{aligned}$$

We can now move the $V(\boldsymbol{x}, t + \mathrm{dt})$ term to the other side, divide by dt and take the limit of $\mathrm{dt} \to 0$ to end up with the continuous time optimality condition for the value function, also called Hamilton-Jacobi Bellman (HJB) equation,

$$\begin{aligned}-\dot{V}(\boldsymbol{x}, t) &= \lim_{\mathrm{dt}\to 0}\frac{V(\boldsymbol{x}, t) - V(\boldsymbol{x}, t + \mathrm{dt})}{\mathrm{dt}}\\ &= r_t(\boldsymbol{x}_t) + \max_{\boldsymbol{u}}\Big(- \boldsymbol{u}_t\boldsymbol{R}_t\boldsymbol{u}_t + (\boldsymbol{f}_t + \boldsymbol{G}_t\boldsymbol{u}_t)^T\boldsymbol{v}_{\boldsymbol{x}} + \frac{1}{2}\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{x}}\boldsymbol{V}_{\boldsymbol{x}\boldsymbol{x}})\Big).\end{aligned}$$

As the HJB contains only quadratic terms of $\boldsymbol{u}_t$, we can obtain the optimal controls by setting the derivative with respect to $\boldsymbol{u}_t$ to zero, i.e.,

$$\boldsymbol{u}_t = \boldsymbol{R}^{-1}\boldsymbol{G}_t^T v_{\boldsymbol{x}}. \tag{2.91}$$

Setting the optimal controls back into the HJB equation yields

$$-\dot{V}(\boldsymbol{x},t) = r_t(\boldsymbol{x}) + \boldsymbol{v}_{\boldsymbol{x}}^T \boldsymbol{f}_t + \boldsymbol{v}_{\boldsymbol{x}}^T \boldsymbol{G}_t \boldsymbol{R}^{-1} \boldsymbol{G}_t^T \boldsymbol{v}_{\boldsymbol{x}} + \tfrac{1}{2}\mathrm{tr}\left(V_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{\Sigma}_x\right), \quad (2.92)$$

the partial differential equation, which has to be satisfied by the optimal value function for the system in Equation (2.84).

Path Integrals. The HJB equation is now transformed to a system of linear partial differential equations by performing an exponential transformation of the optimal value function $\Psi(\boldsymbol{x},t) = \exp(V(\boldsymbol{x},t)/\lambda)$, where λ can be seen as the temperature of the exponential transformation. Under the assumption that the quadratic control cost matrix $\boldsymbol{R}$ is given by the system noise, i.e., $\boldsymbol{R} = \lambda \boldsymbol{\Sigma}_u^{-1}$, the solution for Ψ can be obtained by applying the Feynman-Kac theorem [55] for solving partial differential equations. The solution is given by

$$\Psi(\boldsymbol{x},t) = \int p_{\mathrm{uc}}(\boldsymbol{\tau}|\boldsymbol{x},t) \exp\left(\frac{\sum_{h=t}^{T} r_h(\boldsymbol{x}_h)\mathrm{dt} + r_T(\boldsymbol{x}_T)}{\lambda}\right) d\boldsymbol{\tau}, \quad (2.93)$$

where $p_{\mathrm{uc}}(\boldsymbol{\tau}|\boldsymbol{x},t)$ is the probability of the process using the uncontrolled dynamics $\dot{\boldsymbol{x}}_t = \boldsymbol{f}_t + \boldsymbol{G}_t(\boldsymbol{\epsilon}_t)$, i.e., $\boldsymbol{u}_t = 0$, when starting in state $\boldsymbol{x}$ at time step t [81]. Note that Equation (2.93) is given in its discrete-time form, wherein the discretization time step is again denoted as dt. For more details on the exponential transformation of the value function and the Feynman-Kac theorem we refer to [81]. From the assumption $\boldsymbol{R} = \lambda \boldsymbol{\Sigma}_u^{-1}$, we can also conclude that λ specifies the control costs. The higher we choose the temperature λ of the exponential transformation, the less greedy the exponential transformation will become. This intuition is also reflected in the increasing control costs with increasing λ, and, consequently, the resulting solution will become more similar to the uncontrolled process. In practice, the assumption for the control cost matrix $\boldsymbol{R}$ is rather limiting, as we are not free in our choice of the control costs.

We will further define $S(\boldsymbol{\tau}|\boldsymbol{x}_t,t)$ to be the path integral of trajectory $\boldsymbol{\tau}$ starting at time step t in state $\boldsymbol{x}$ which is given as

$$S(\boldsymbol{\tau}|\boldsymbol{x}_t,t) = r_T(\boldsymbol{x}_T) + \sum_{h=t}^{T} r_h(\boldsymbol{x}_h)\mathrm{dt} + \log p_{\mathrm{uc}}(\boldsymbol{\tau}|\boldsymbol{x},t). \quad (2.94)$$

Hence, the path integral of a trajectory for time step t is given by the reward to come plus a logarithmic punishment term which renders less likely trajectories less attractive. Due to the assumption of $\boldsymbol{R} = \lambda \Sigma_u^{-1}$, the return and the probability of a trajectory can be treated in a unified way. The optimal controls $\boldsymbol{u}_t$ can be obtained by

$$\boldsymbol{u}_t = -\boldsymbol{R}^{-1}\boldsymbol{G}_t^T \boldsymbol{v}_{\boldsymbol{x}} = \lambda \boldsymbol{R}^{-1}\boldsymbol{G}_t^T \frac{\partial/\partial_{\boldsymbol{x}}\Psi(\boldsymbol{x},t)}{\Psi(\boldsymbol{x},t)}. \tag{2.95}$$

Determining the term $\left(\partial/\partial_{\boldsymbol{x}}\Psi(\boldsymbol{x},t)\right)/\Psi(\boldsymbol{x},t)$ and setting this term into Equation (2.95), yields [81]

$$\boldsymbol{u}_t = \int p_{\mathrm{co}}(\boldsymbol{\tau}|\boldsymbol{x}_t,t)\boldsymbol{u}_L(\boldsymbol{\tau},t)d\boldsymbol{\tau}, \tag{2.96}$$

where

$$p_{\mathrm{co}}(\boldsymbol{\tau}|\boldsymbol{x}_t,t) = \frac{\exp\left(S(\boldsymbol{\tau}_t|\boldsymbol{x}_t,t)/\lambda\right)}{\int \exp\left(S(\boldsymbol{\tau}_t|\boldsymbol{x}_t,t)/\lambda\right)d\boldsymbol{\tau}} \tag{2.97}$$

$$\propto p_{\mathrm{uc}}(\boldsymbol{\tau}_t|\boldsymbol{x}_t,t)\exp\left(\sum_{h=t}^{T} r_h(\boldsymbol{x}_h)\mathrm{dt} + r_T(\boldsymbol{x}_T)\right) \tag{2.98}$$

and

$$\boldsymbol{u}_L(\boldsymbol{\tau}_t) = \boldsymbol{R}^{-1}\boldsymbol{G}_t\left(\boldsymbol{G}_t\boldsymbol{R}^{-1}\boldsymbol{G}_t^T\right)^{-1}\boldsymbol{G}_t\boldsymbol{\epsilon}_t. \tag{2.99}$$

We will denote the distribution $p_{\mathrm{co}}(\boldsymbol{\tau}|\boldsymbol{x}_t,t)$ as controlled process distribution, as it denotes the trajectory distribution connected to the optimal (transformed) value function Ψ_t. We observe that the controlled process distribution is represented as a soft-max distribution which has the path integrals $S(\boldsymbol{\tau}|\boldsymbol{x},t)$ of the trajectory in its exponent. Alternatively, $p_{\mathrm{co}}(\boldsymbol{\tau}|\boldsymbol{x}_t,t)$ can also be written as the uncontrolled process distribution $p_{\mathrm{uc}}(\boldsymbol{\tau}|\boldsymbol{x}_t,t)$ that is weighted by the exponentially transformed reward to come.

The action $\boldsymbol{u}_L(\boldsymbol{\tau},t)$ is denoted as correction action of trajectory $\boldsymbol{\tau}$. It is defined as the action which follows the trajectory $\boldsymbol{\tau}$ while minimizing the immediate control costs [81] at time step t. The term $\boldsymbol{\epsilon}_t$ is the noise term applied at time step t for trajectory $\boldsymbol{\tau}$. The matrix $\boldsymbol{R}^{-1}\boldsymbol{G}_t\left(\boldsymbol{G}_t\boldsymbol{R}^{-1}\boldsymbol{G}_t^T\right)^{-1}\boldsymbol{G}_t$ projects the applied noise term into the null-space of the control matrix $\boldsymbol{G}_t$, and, hence, eliminates the unnecessary part of the control action $\boldsymbol{u}$.

By combining Equations (2.99) and (2.96), we can summarize that the optimal control law is given in the form of a soft-max distribution $p_{\mathrm{co}}(\boldsymbol{\tau}|\boldsymbol{x},t)$, which weights relevant parts of the noise term $\boldsymbol{\epsilon}_t$ according to their path integrals $S(\boldsymbol{\tau},t)$. The main advantage of path integrals is that the optimal action can be obtained by performing Monte-Carlo roll-outs instead of applying dynamic programming. As in the REPS approach, the maximum-operation, which is typically needed to obtain the optimality of the action, is replaced by a soft-max operator, which is easier to perform. In the following, the condition on the start state $\boldsymbol{x}$ of the trajectories will be dropped in order to make the computations feasible. However, this simplification might again add a bias to the resulting path integral approach. The effect of this bias still has to be examined.

Iterative Path Integral Control. In practice, we have to sample from the uncontrolled process to solve the integral in Equation (2.95) over the trajectory space. However, this sampling process can be inefficient for high-dimensional systems, wherein high number of samples is needed to obtain an accurate estimate. The number of required samples can be reduced by using an iterative approach. At each iteration k, we only compute the optimal change $\delta_k \boldsymbol{u}_t$ in the action for time step t. Subsequently, the mean action $\boldsymbol{u}_{k,t}$ for time step t is updated by $\boldsymbol{u}_{k,t} = \boldsymbol{u}_{k-1,t} + \delta_k \boldsymbol{u}_t$. Such procedure allows for the use of a smaller system noise $\boldsymbol{\Sigma}_{\boldsymbol{u}}$ for exploration, and hence, we can search for a locally optimal improvement of the options. As we now search for an optimal change of the action $\delta \boldsymbol{u}_t$, the current estimate $\boldsymbol{u}_{k,t}$ of the action is subsumed in the drift-term of the dynamics, i.e., $\tilde{\boldsymbol{f}}_{k,t} = \boldsymbol{f}_t + \boldsymbol{G}_t \boldsymbol{u}_{k,t}$. However, the explicit dependence of $\boldsymbol{u}_{k,t}$ from the current state is ignored in this iterative formulation. Note that the path integral approach only estimates the mean control action of the policy and not the variance. Exploration is solely performed by the user-specified uncontrolled dynamics.

Policy Improvements by Path Integrals. The Policy Improvement by Path Integrals (PI^2) algorithm [81] applies the path integral theory to the problem of learning Dynamic Movement Primitives (DMPs) as policy representations. In this section, we restrict ourselves to learning a DMP for a single joint, for multiple joints, learning can be performed by applying the discussed update rules for each joint separately. The path integral theory can be applied directly to DMPs by treating the DMP parameter vector $\boldsymbol{w}$ as the control action for the dynamical system

$$\ddot{y} = \underbrace{\tau^2 \alpha_y(\beta_y(g-y) - \dot{y})}_{f_t} + \underbrace{\tau^2 \boldsymbol{\phi}_t}_{\boldsymbol{G}_t}{}^T (\boldsymbol{w}_t + \boldsymbol{\epsilon}_t)\,, \tag{2.100}$$

which defines the DMP trajectory. Note that, as in the PoWER approach [38], we have assumed to use a different parameter vector $\boldsymbol{w}_t$ in each time step. The drift term f_t is given by the spring and damper system of the DMP and the control matrix G_t is given by the basis functions $\boldsymbol{\phi}_t$ of the DMP. Since we apply the exploration noise $\boldsymbol{\epsilon}_t$ at each time step to the parameter vector $\boldsymbol{w}_t$, the exploration policy is given by $\mathcal{N}(u_t|f_t, H_t)$ with $H_t = \boldsymbol{\phi}_t^T \boldsymbol{\Sigma}_{\boldsymbol{w}} \boldsymbol{\phi}_t$.

In order to define the path-integral update rule, we need to compute the path integral $S(\boldsymbol{\tau}|t)$ for a given trajectory. This step requires knowledge of the uncontrolled trajectory distribution $p_{\mathrm{uc}}(\boldsymbol{\tau}|t)$, and, hence, knowledge of the system model. However, if we assume the rewards r_t to depend only on the state of the DMP, and not on the real state of the robot or its environment, i.e., $r_t(\boldsymbol{x}_t) = r_t(y_t, \dot{y}_t)$, the uncontrolled dynamics $p_{\mathrm{uc}}(\boldsymbol{\tau}|\boldsymbol{x}_t) = \prod_{t=0}^{T} p(y_t, \dot{y}_t | y_{t-1}, \dot{y}_{t-1})$ are straightforward to compute. The path integral can be rewritten as

$$S(\boldsymbol{\tau}, t) = r_T(\boldsymbol{x}_T) + \sum_{l=t}^{T-1} r_l(\boldsymbol{x}_l) + (\boldsymbol{w}_l + \boldsymbol{M}_l \epsilon_l)^T \boldsymbol{R}(\boldsymbol{w}_l + \boldsymbol{M}_l \epsilon_l))\,, \tag{2.101}$$

where $\boldsymbol{M}_l = \boldsymbol{\phi}_l H_l^{-1} \boldsymbol{\phi}_l^T$ [81]. The new parameter vector $\boldsymbol{w}_{\mathrm{new},t}$ for time step t is computed by the iterative path integral update rule. Using Equation (2.95) for the optimal control action yields

$$\boldsymbol{w}_{\mathrm{new},t} = \boldsymbol{w}_t + \int p_{\mathrm{co}}(\boldsymbol{\tau}_t) M_t \boldsymbol{\epsilon}_t d\boldsymbol{\tau}, \tag{2.102}$$

where $p_{\mathrm{co}}(\boldsymbol{\tau}_t)$ is given by the soft-max distribution in Equation (2.98). So far, we used a different parameter vector w_t for each time step.

However, in practice, we can use only a single parameter vector $\boldsymbol{w}$ for one episode, and, thus, we need to average over the parameter updates for all time-steps. The average update is computed by weighting each parameter vector $\boldsymbol{w}_{\text{new},t}$ with the number of time steps to go. Additionally, the update for the j-th dimension of the parameter vector $\boldsymbol{w}$ for time step t is weighted by the activation of the j-th feature function $[\boldsymbol{\phi}_t]_j$,

$$
\begin{aligned}
[\boldsymbol{w}_{\text{new}}]_j &= \frac{\sum_{t=0}^{T-1}(T-t)\,[\boldsymbol{\phi}_t]_j\,[\boldsymbol{w}_{\text{new,t}}]_j}{\sum_{t=0}^{T-1}(T-i)\,[\boldsymbol{\phi}_t]_j} \\
&\approx [\boldsymbol{w}_{\text{old}}]_j + \sum_{t=0}^{T-1}\sum_{i=1}^{N} p_{\text{co}}\big(\boldsymbol{\tau}_t^{[i]}\big)\,d_{t,j}\left[\boldsymbol{\epsilon}_t^{[i]}\right]_j
\end{aligned}
$$

with

$$
d_{t,j} = (T-t)\,[\boldsymbol{\phi}_t]_j \,/ \Big(\sum_{t'=0}^{T}(T-t')\,[\boldsymbol{\phi}_{t'}]_j\Big). \tag{2.103}
$$

Equation (2.103) defines the update rule of the original PI^2 algorithm given in [81]. We observe that PI^2 fits into our categorization of using a step-based policy evaluation strategy, which uses the future reward in the current trajectory plus a punishment term for unlikely trajectories as the evaluation. Similar to the PoWER [38] algorithm, this evaluation is used by the soft-max policy to obtain a weighting for each of the samples.

Episode-Based PI^2. The PI^2 algorithm has also been used in the episode-based formulation [77], which also also has been used for updating the exploration strategy. The basic PI^2 algorithm does not update its exploration strategy and has to rely on the uncontrolled process dynamics, which is typically set by the user. However, the noise-variance of the uncontrolled process dynamics has a large impact on the learning performance and, hence needs to be chosen appropriately. To automatically estimate the exploration strategy, the PI^2 algorithm was reformulated in the episode-based policy search formulation and the condition that the noise covariance $\boldsymbol{\Sigma}_{\boldsymbol{u}}$ needs the match the control costs matrix $\boldsymbol{R}$ was explicitly ignored. Due to the episode-based formulation, the policy can be directly estimated in parameter space. Instead of using the uncontrolled process for exploration, the previously estimated policy is used for exploration. Furthermore, the log-term for the uncontrolled dynamics $p_{\mathrm{uc}}(\boldsymbol{\tau}|\boldsymbol{x}_t)$ in the path integral $S(\boldsymbol{\tau})$ is neglected as this term does not seem to improve the performance. Consequently, in the episode-based formulation the returns $R^{[i]}$ are directly used as path integrals. The covariance matrix was updated by a weighted maximum likelihood estimate, where the soft-max distribution $p_{\mathrm{co}}(\boldsymbol{\tau}_t)$ was used as weighting. The resulting episode-based PI^2 is a simplified version of the original PI^2 algorithm, but shows an improved learning performance.

Relation to Information-Theoretic and EM Approaches. Despite that information theoretic and EM approaches were developed from different principles than the path integral approach, all these approaches share similar characteristics.

The episode-based formulation of PI^2 shares many similarities with the episode-based REPS formulation. By pulling the $\log p_{\mathrm{uc}}(\boldsymbol{\tau}_t|\boldsymbol{x}_t)$ term outside the exponent, we realize that $p_{\mathrm{co}}(\boldsymbol{\tau}|\boldsymbol{x}_t)$ shares a similar soft-max structure as the closed form solution in REPS for $\pi(\boldsymbol{u}|\boldsymbol{x})$. In REPS, the trajectory distribution $q(\tau)$ of the old policy is used instead of the uncontrolled process distribution $p_{\mathrm{uc}}(\boldsymbol{\tau}_t|\boldsymbol{x}_t)$. A similar strategy is emulated by using the iterative path integrals update, where the mean of the exploration policy is updated, but the exploration noise is always determined by the uncontrolled process. While REPS is designed to be an iterative algorithm, the iterative sampling process of PI^2 needs to be motivated from a heuristic view point.

We also observe that the temperature parameter λ in path integral control corresponds to the Lagrangian parameter η in REPS. This parameter is automatically set in REPS according to the relative entropy bound, while for path integral control, λ is set by heuristics. The path integral approach also relies on the assumption that the control cost matrix $\boldsymbol{R}$ is predefined as $\lambda\boldsymbol{R}^{-1} = \boldsymbol{\Sigma}_u$. Dropping this assumption results in a more efficient algorithm [77], but the theoretical justifications for such an algorithm are also lost. Similar update rules emerge naturally for the REPS algorithm without the need for heuristics. However, REPS has so far only been introduced for the episode-based policy search learning formulation, and, hence, makes inefficient usage of the available trajectory samples $\boldsymbol{\tau}^{[i]}$. PI^2 has been derived from a step-based formulation, and, hence, might have advantages over REPS in some applications.

The original step-based version of the PI^2 algorithm is also closely related to the PoWER algorithm. If we also use an exponential transformation of the reward for PoWER, the policy updates are essentially the same. While PoWER uses a weighted maximum likelihood update to obtain the new policy, PI^2 averages the update rules for the single time steps.

2.5 Real Robot Applications with Model-Free Policy Search

In this section, we present selected results for model-free policy search in the area of robotics. These experiments include Baseball, Ball-In-The-Cup, Dart-Throwing, Pan-Cake Flipping and Tetherball. All experiments have been conducted with dynamic movement primitives as policy representation. In all applications, learning with DMPs takes place in two phases [38]. In the first phase, imitation learning is used to reproduce recorded trajectories. Subsequently, reinforcement learning is used to improve upon the imitation. The use of imitation learning to initialize the learning process allows for the incorporation of experts knowledge and can considerably speed up the learning process. Most experiments have been performed with a specific algorithm in mind that was at the time of the experiment state of the art. However, more recent approaches such as REPS or PI^2 would also have worked in most setups[4].

2.5.1 Learning Baseball with eNAC

In the baseball experiment, a Sarcos Master Arm was used to hit a soft baseball placed on a T-stick such that it flies as far as possible [61]. This game is also called T-Ball and used to teach children how to hit

[4] While REPS also works for contextual policy search, PI^2 was so far not extended to the multi-task setup.

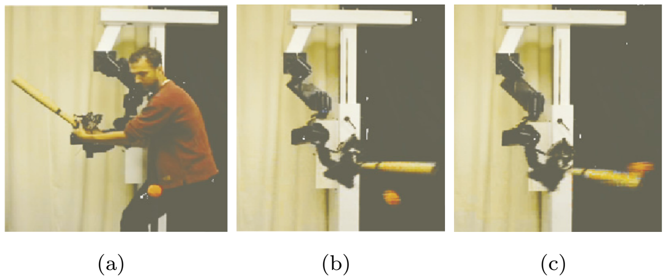


Fig. 2.6 Learning a baseball swinging movement to hit a ball placed on a T-stick [61]. (a) Obtaining an initial solution by imitation learning. (b) Initial solution replayed by the robot. The robot misses the ball. (c) Movement learned with the eNAC algorithm after 300 roll-outs. The robot learned to hit the ball robustly.

a baseball. The robot had seven degrees of freedom (DoF) and ten basis functions where used for each DoF. Only the shape parameters of the DMP were adapted during learning, resulting in a 70-dimensional weight vector $\boldsymbol{w}$. The reward function was given by

$$R(\boldsymbol{\tau}) = c_1 p_x - c_2 \sum_{t=0}^{T-1} \ddot{\boldsymbol{q}}_t^T \ddot{\boldsymbol{q}}_t,$$

where p_x is the distance the ball traveled and c_1 and c_2 are constants to weight the objective of hitting the ball versus minimizing the energy consumption. An initial solution was obtained by imitation learning, see Figure 2.6(a). However, the robot failed to exactly reproduce the behavior and missed the ball, Figure 2.6(b). The behavior could be improved by employing the episodic Natural Actor Critic algorithm. After 200 to 300 trials of learning, the robot was able the reliably hit the ball, see Figure 2.6(c).

2.5.2 Learning Ball-in-the-Cup with PoWER

The PoWER algorithm was used in [38] to learn the game ‘Ball-in-the-Cup’. The used robot platform was a Barrett WAM robot arm with seven degrees of freedom. In total, the authors selected 31 basis functions to learn the task. The shape parameters of the DMP were learned as well as the variance σ_i^2 for each basis function. All degrees of freedom are perturbed separately but share the reward, which is zero except for the time step t_c, where the ball passes the cup rim in a downward direction. A stereo vision system was used to track the position $\boldsymbol{b} = [x_b, y_b, z_b]^T$ of the ball. This ball position was used for determining the reward, but not for feedback during the motion. The reward at time-step t_c was given by

$$r_{t_c} = \exp(-\alpha(x_c - x_b) - \alpha(y_c - y_b)),$$

where x_c and y_c are the x and y-coordinates of the cup and x_b and y_b the coordinates of the ball at this time step. The parameter α is a scaling parameter which is set to 100. The exp function is used to transform the squared distance of the ball to the cup into an improper probability distribution, as PoWER requires this type of reward function. The policy was initialized with imitation learning. After 75 trials, the robot could reliably catch the ball with the cup. The resulting learning progress is illustrated in Figure 2.7, which shows the position of the ball closest to the cup for a different number of roll-outs. This experiment was extended in [35] to include feedback for the position of the ball. Here, a DMP was used to learn the expected trajectory of the ball. For each degree of freedom, an additional PD-controller was learned which is added to the forcing function of the DMP. The PD controller takes the deviation of the current ball position and velocity to the expected ball position and velocity as input. This experiment was only conducted in simulation, the initial position and velocity of the ball were perturbed randomly. The DMP, including the additional feedback terms, contained 91 parameters which were optimized by the PoWER approach. Due to the high variance in the initial state of the ball, the PoWER algorithm now converged after 500 to 600 episodes.

2.5.3 Learning Pan-Cake Flipping with PoWER/RWR

In [40], the PoWER algorithm was used to learn to flip a pan-cake with a Barrett WAM. As underlying policy representation, an extension of the DMP approach was used where the spring-damper system of the DMP was also modeled as time dependent, and, additionally, the

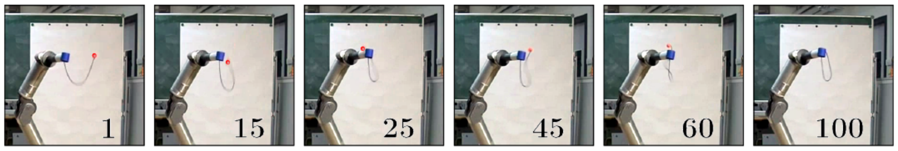


Fig. 2.7 Learning the game 'Ball-in-the-cup' [38]. The robot has to learn how to catch the ball which is connected with a string to the cup. The initial policy is learned by imitation and shown on the left. The figures show the minimum distance of the ball to the cup with an increasing number of roll-outs. After 75 episodes learning with the PoWER algorithm, the robot was able to catch the ball reliably.

spring-damper system for each dimension were coupled[5]. The DMP were defined in task space, i.e., in the 3-D Cartesian coordinates of the end-effector position. The coupling terms of the spring damper system as well as the goal attractor where learned. The reward function was composed of three terms: The rotation error of the pan-cake before landing in the pan, the distance of the pan-cake to the center of the pan when catching the pan-cake, and the maximum height the pan-cake reached. The authors state that they used the PoWER algorithm, however, we would rather classify the used algorithm as Reward Weighted Regression as it uses an episode-based evaluation and exploration strategy. The intermediate steps of the episode, a key characteristics of the PoWER algorithm, are neglected. The robot was able to learn the task after 50 roll-outs. A successful pan-cake-flipping episode is indicated in Figure 2.8. The robot also learned how to adjust the stiffness of the joints due to the time-varying spring damper system of the DMP. When catching the pan-cake, the robot increased the compliance of the arm such that the arm can freely move down and prevents the pan-cake from bouncing out of the pan.

2.5.4 Learning Dart Throwing with CRKR

In [37], CRKR was used to learn dart throwing with a robot. The robot had to hit different locations on a dart board. The dart is placed

[5] Coupling is typically achieved with the phase variable in the DMP framework. However, using coupling also for the feedback terms of the spring damper system allows for the use of correlated feedback.

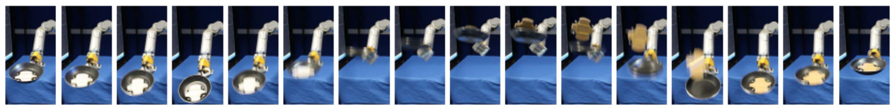


Fig. 2.8 Learning the pan-cake-flipping task [40]. The robot has to flip a pan-cake such that it rotates 180 degrees in the air, and, subsequently, the robot has to catch it again. The figure illustrates the learned movement after 50 roll-outs. The authors applied the RWR algorithm to learn the movement.

on a launcher attached to the end-effector and held there by stiction. CRKR learns an upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$ where the context $\boldsymbol{s}$ is given by the two-dimensional location of the target on the dart board. The parameters $\boldsymbol{\theta}$ contained the the goal-attractor $\boldsymbol{g}$, the final velocity $\dot{\boldsymbol{g}}$ and the time scale constant τ of the DMP. The shape parameters $\boldsymbol{w}$ of the DMP were obtained via imitation learning and fixed during learning. The reward function was given by the negative distance of the impact position $\boldsymbol{d}$ of the dart to the desired position $\boldsymbol{x}$ and an additional punishment term for fast movements, i.e.,

$$R(\boldsymbol{\theta}, \boldsymbol{x}) = -10||\boldsymbol{d} - \boldsymbol{x}|| - \tau,$$

where τ is the time-scale constant of the DMP, which controls the velocity of the movement. The experiment was conducted on three real-robot platforms, the Barrett WAM, the humanoid robot CBi and the Kuka CR 6. For all robot platforms, CRKR was able to learn a successful throwing movement and hit the desired target within the range of ± 10cm after 200 to 300 trials, which was within the reproduction accuracy of the robots. The learned movement for the CBi is illustrated in Figure 2.9.

2.5.5 Learning Table Tennis with CRKR

CRKR was also used to learn to return table-tennis balls [37]. The authors used a Barrett WAM with seven DoF, which was mounted on the ceiling. The hitting movement for table-tennis was decomposed into three dynamic movement primitives. In the first phase, the robot swings back. The second movement primitive is then used to hit the ball while the third primitive is used to smoothly go back the initial position. The

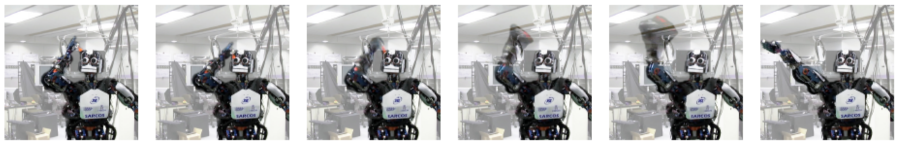


Fig. 2.9 Learning the dart throwing task [37]. CRKR was used to generalize a dart throwing movement to different target locations. After 300 episodes, the humanoid robot CBi was able to reliably throw the dart within a range of ±10cm distance to the target. CRKR adapted the meta-parameters of the DMP which included the goal position $\boldsymbol{g}$ and goal velocity $\dot{\boldsymbol{g}}$ for each joint and the time scaling constant of the DMP.

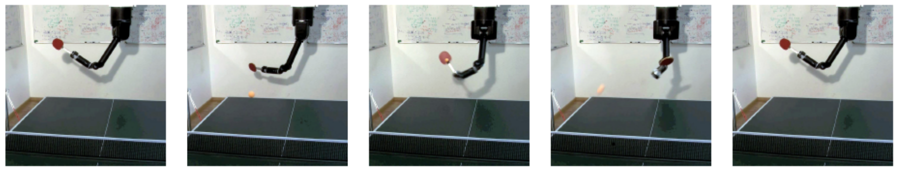


Fig. 2.10 Learning to hit a table tennis ball in the air [37]. CRKR was used to learn an upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$, where the context $\boldsymbol{s}$ was given by the position and velocity of the ball when it passes the net. The policy parameters $\boldsymbol{\theta}$ included the final positions $\boldsymbol{g}$ and velocities $\dot{\boldsymbol{g}}$ of the joint as well as a waiting time when to start the hitting movement. The robot was able to hit the ball for different configurations of the incoming ball and improved its success rate for hitting the ball from 30% to 80%.

meta-parameters of the second primitive, including the final position $\boldsymbol{g}$ and final velocities $\dot{\boldsymbol{g}}$ of the movement are learned. Additionally, a timing parameter t_{hit} is learned that controls the the transition from swing-back to hitting primitive. Hence, the resulting parameter vector $\boldsymbol{\theta}$ included 15 parameters. The reward function was given by the negative distance of the racket to the ball at the estimated hitting time point t_{hit}. The upper-level policy was initialized with five successful examples obtained from another player. The robot was able to increase its success rate to hit the ball from an initial 30% to 80% after 100 roll-outs. A successful hitting movement is illustrated in Figure 2.10.

2.5.6 Learning Tetherball with HiREPS

The HiREPS algorithm was used to learn several solutions for the game of Tetherball [18]. A ball was attached to a string which hung from the

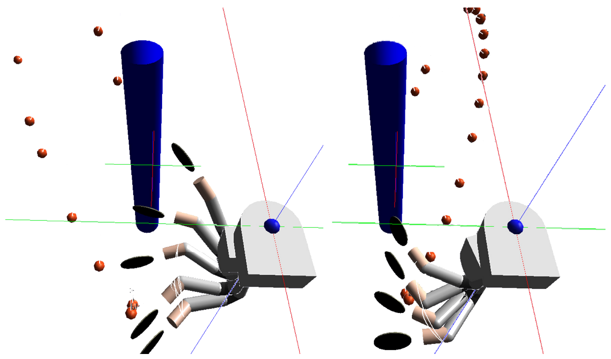


Fig. 2.11 Learning the Tetherball task [18]. The task is to hit a ball such that it winds around the pole. There are two solutions, hitting the ball to the left and hitting the ball to the right. HiREPS was able to learn both solutions in 300 trials. The figure shows both learned movement in simulation.

ceiling. A pole is placed in front of the robot. The task of the robot is to wind the ball around the pole. To achieve the task, the robot first needed to hit the ball once to displace it from its resting pose, and, subsequently, hit it again to arc it around the pole. This task has two solutions: wind the ball around the pole clockwise or counter-clockwise.

The movement was decomposed into a swing-in motion and a hitting motion. For both motions, the shape parameters $\mathbf{w}$ were extracted by kinesthetic teach-in. Both motions where represented by a single set of parameters and the parameters for the two DMPs were jointly learned. For both movements, the final positions $\boldsymbol{g}$ and velocities $\dot{\boldsymbol{g}}$ of all seven joints were learned. Additionally, the waiting time between both movements was included in the parameters. This task setup results in a 29-dimensional parameter space.

The reward was determined by the speed of the ball when the ball winds around the pole, where winding around the pole was defined as the ball passing the pole on the opposite side from the initial position. After 300 roll-outs HiREPS was able to learn both solutions within one learning trial as shown in Figure 2.11.

3

Model-based Policy Search

Model-free policy search methods as described in Section 2 are inherently based on sampling trajectories $\boldsymbol{\tau}^{[i]}$ using the robot to find good policies π^*. Sampling trajectories is relatively straightforward in computer simulation. However, when working with mechanical systems, such as robots, each sample corresponds to interacting directly with the robot, which often requires substantial experimental time and causes wear and tear in non-industrial robots. Depending on the task, it can either be easier to learn a model or to learn a policy directly. Model-based policy search methods attempt to address the problem of sample inefficiency by using observed data to learn a forward model of the robot's dynamics. Subsequently, this forward model is used for *internal* simulations of the robot's dynamics, based on which the policy is learned.

Model-based policy search algorithms typically assume the following set-up: The state $\boldsymbol{x}$ evolves according to the Markovian dynamics

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}\,, \quad \boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)\,, \tag{3.1}$$

where f is a nonlinear function, $\boldsymbol{u}$ is a control signal (action), and $\boldsymbol{w}$ is additive noise, often chosen to be i.i.d. Gaussian. Moreover, an episodic set-up is considered where the robot is reset to an initial state $\boldsymbol{x}_0$ after executing a policy. The initial state distribution $p(\boldsymbol{x}_0)$ is often given by a Gaussian distribution $\mathcal{N}\big(\boldsymbol{x}_0 \,|\, \boldsymbol{\mu}_0^x, \boldsymbol{\Sigma}_0^x\big)$. Furthermore, we consider finite horizon problems, i.e., the policy-search objective is to find a parametrized policy $\pi_{\boldsymbol{\theta}}^*$ that maximizes the expected long-term reward

$$\pi_{\boldsymbol{\theta}}^* \in \arg\max_{\pi_{\boldsymbol{\theta}}} J_{\boldsymbol{\theta}} = \arg\max_{\pi} \sum_{t=1}^{T} \gamma^t \mathbb{E}[r(\boldsymbol{x}_t, \boldsymbol{u}_t)|\pi_{\boldsymbol{\theta}}]\,, \quad \gamma \in [0,1]\,, \quad (3.2)$$

where r is an immediate reward signal, γ a discount factor, and the policy $\pi_{\boldsymbol{\theta}}$ is parametrized by parameters $\boldsymbol{\theta}$. Therefore, finding $\pi_{\boldsymbol{\theta}}^*$ in Equation (3.2) is equivalent to finding the corresponding optimal policy parameters $\boldsymbol{\theta}^*$.

For some problems, model-based RL methods have the promise of requiring fewer interactions with the robot than model-free RL by learning a model of the transition mapping in Equation (3.1), while efficiently generalizing to unforeseen situations using a model learned from observed data [6].

The general idea of model-based RL is depicted in Figure 3.1. The learned model is used for internal simulations, i.e., predictions about how the real robot and its environment would behave if it followed the current policy. Based on these internal simulations, the quality of the policy is evaluated using Equation (3.2) and improved accordingly. Subsequently, the updated policy is again evaluated using Equation (3.2) and improved. This policy evaluation/improvement loop terminates when the policy is learned, i.e., it no longer changes and a (local) optimum is attained. Once a policy is learned, it is applied to the robot and a new data set is recorded. Combined with previously collected data, the data set is used to update and refine the learned dynamics model. In theory, this loop continues forever. Note that, only the *application* of the policy requires interacting with the robot; internal simulations and policy learning only use the learned computer model of the robot dynamics.

While the idea of using models in the context of robot learning is well-known since Aboaf's work in the 1980s [2], it has been limited by its strong dependency on the quality of the learned model, which becomes also clear from Figure 3.1: The learned policy is inherently

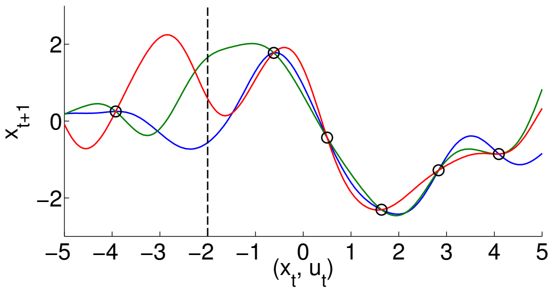


Fig. 3.2 Model errors. In this example, six function values have been observed (black circles). Three functions are shown that could have generated these observations. Any single function (e.g., the maximum likelihood function) produces more or less arbitrary predictions in regions with sparse training data, one example location of which is shown by the dashed line. Instead of selecting a single function approximator, we describe our uncertainty about the underlying function by a probability distribution to be robust to such model errors.

based on internal simulations using the learned model. When the model exactly corresponds to the true dynamics of the robot, sampling from the learned model is equivalent to sampling from the real robot.

However, in practice, the learned model is *not* exact, but only a more or less accurate approximation to the real dynamics. For example, in regions where the training data is sparse, the quality of the learned model can be insufficient as illustrated in Figure 3.2. There are multiple plausible functions that could have generated the observed function values (black circles). In regions with sparse training data, the models and their predictions differ significantly. Any single model

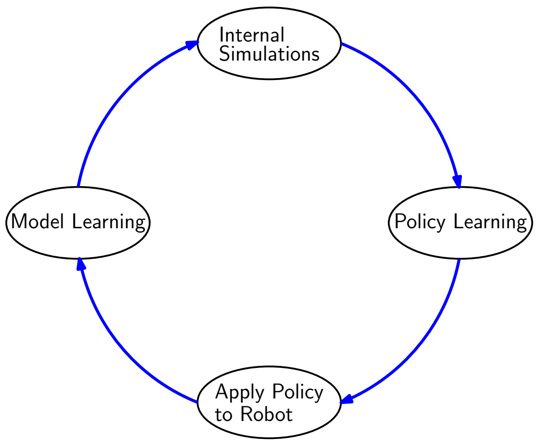


Fig. 3.1 General loop in model-based RL: The learned model is used for internal simulations (mental rehearsal) based on which the policy is improved. The learned policy is then applied to the robot. The data from this interaction is used to refine the model, and the loop continues until the model and the learned policy converge.

leads to overconfident predictions that, in turn, can result in control strategies that are not robust to model errors. This behavior can have a drastic effect in robotics, for example, resulting in the estimation of negative masses or negative friction coefficients. These implausible effects are often exploited by the learning system since they insert energy into the system, causing the system to believe in "perpetuum mobiles". Therefore, instead of selecting a single model (e.g., the maximum likelihood model), we should describe our uncertainty about the latent underlying function f by a probability distribution $p(f)$ to be robust to such model errors [72, 7, 20]. By taking model uncertainty into account, the perpetuum mobile-effect is substantially less likely.

Since the learned policy inherently relies on the quality of the learned *forward model*, which essentially serves as a simulator of the robot, model errors can not only cause degraded policies, but they also often drastically bias the learning process. Hence, the literature on model-based policy search largely focuses on model building, i.e., explaining what kind of model is used for the forward dynamics and how it is trained.

Approaches to Dealing with Uncertain Models. Dealing with inaccurate dynamics models is one of the biggest challenges in model-based RL since small errors in the model can lead to large errors in the policy [6]. Inaccuracies might stem from an overly restrictive model class or from the lack of sufficiently rich data sets used for training the models, which can lead to under-modeling the true forward dynamics. Moreover, system noise typically adds another source of uncertainty.

Typically, a certainty-equivalence assumption is made[1], and the maximum likelihood model is chosen for planning [72, 7]. However, this certainty-equivalence assumption is violated in most interesting cases and can lead to large policy errors. Moreover, as mentioned already in [2], many approaches obtain derivatives of the expected return by back-propagating derivatives through learned forward models of the system. This step is particularly prone to model errors since gradient-based

[1] It is assumed that the optimal policy for the learned model corresponds to the optimal policy for the true dynamics. Uncertainty about the learned model is neglected.

optimizers improve the policy parameters along their gradients. The learned policy needs to be robust to compensate for model errors such that it results in good performance when applied to the real system. Learning faithful dynamics models is crucial for building robust policies and remains one of the biggest challenges in model-based policy search.

In [45], the authors model unknown error dynamics using receptive-field weighted regression [70]. Explicit modeling of unknown disturbances leads to increased robustness of the learned controllers. The idea of designing controllers in the face of inaccurate (idealized) forward models is closely related to robust control in classical robotics. Robust control aims to achieve guaranteed performance or stability in the presence of (typically bounded) modeling errors. For example, $\mathcal{H}_\infty$ loop-shaping [44] guarantees that the system remains close to its expected behavior even if (bounded) disturbances enter the system. In adaptive control, parameter uncertainties are usually described by unbounded probability distributions [5]. Model parameter uncertainty is typically not used in designing adaptive control algorithms. Instead, the estimates of the parameters are treated as the true ones [90]. An approach to designing adaptive controllers that do take uncertainty about the model parameters into account is stochastic adaptive control [5]. When reducing parameter uncertainty by probing, stochastic adaptive control leads to the principle of dual control [26]. Adaptive dual control has been investigated mainly for linear systems [90]. An extension of dual adaptive control to the case of nonlinear systems with affine controls was proposed in [25]. A minimum-variance control law is obtained, and uncertainty about the model parameters is penalized to improve their estimation, eliminating the need of prior system identification.

RL approaches that explicitly address the problem of inaccurate models in robotics have only been introduced recently [72, 7, 45, 51, 53, 1, 34, 21, 20]. For instance, in [1], the key idea is to use a real-life trial to evaluate a policy, but then use a crude model of the system to estimate the derivative of the evaluation with respect to the policy parameters (and suggest local improvements). In particular, the suggested algorithm iteratively updates the model f according to $f^{[i+1]}(\boldsymbol{x}_t, \boldsymbol{u}_t) = f^{[i]}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{x}_{t+1}^{[i]} - f^{[i]}(\boldsymbol{x}_t^{[i]}, \boldsymbol{u}_t^{[i]})$, where t indexes time.

Here, $\boldsymbol{x}_t^{[i]}, \boldsymbol{u}_t^{[i]}$ are taken from an observed trajectory. It follows that the updated model $f^{[i+1]}$ predicts the observed trajectory exactly, i.e., $f^{[i+1]}(\boldsymbol{x}_t^{[i]}, \boldsymbol{u}_t^{[i]}) = \boldsymbol{x}_{t+1}^{[i]}$. The algorithm evaluates the policy gradients along the trajectory of states and controls in the real system. In contrast, a typical model-based approach evaluates the derivatives along the trajectory predicted by the model, which does not correspond to the trajectory of the real system when the model is inexact. Note that the approach in [1] does not directly generalize to stochastic transition dynamics or systems with hidden states. Moreover, an approximate parametric model of the underlying dynamics need to be known in advance.

Major Challenges in Model-based Policy Search. There are three general challenges that need to be addressed in model-based policy search methods: what model to learn, how to use the model for long-term predictions, and how to update the policy based on the long-term predictions. These three challenges correspond to the three components in Figure 3.1 that do not require physical interaction with the robot: model learning, internal simulations, and policy learning.

In Section 3.1, we give a brief overview of two models that are frequently used in model-based policy search [7, 51, 53, 34, 21, 20], locally weighted (Bayesian) regression (LWBR) and Gaussian processes (GPs) [65]. In Section 3.2, we discuss two general methods of how to use these models for long-term predictions: stochastic inference, i.e., sampling, and deterministic approximate inference. In Section 3.3, we briefly discuss multiple options of updating the policy.

After having introduced these general concepts, in Section 3.4, we discuss model-based policy search algorithms that combine the learned models and inference algorithms shown in Table 3.1. We focus on the episodic case, wherein the start state distribution $p(\boldsymbol{x}_0)$ is known. In particular, we detail four model-based policy search methods. First, we start with PEGASUS, a general concept for efficient trajectory sampling in stochastic MDPs for a given model [52]. Second, we present two ways of combining PEGASUS with LWBR for learning robust controllers to fly helicopters [7, 51, 53]. Third, we present an approach

Table 3.1 Model-based policy search approaches grouped by the learned model and the method of generating trajectories. Due to the simplicity of sampling trajectories, most policy search methods follow this approach. Deterministic trajectory predictions can only be performed in special cases where closed-form approximate inference is possible. Although they are mathematically more involved than trajectory sampling, they do not suffer from large variances of the samples. Moreover, they can allow for analytic gradient computations, which is crucial in the case of hundreds of policy parameters.

	Trajectory Prediction	
Learned Forward Model	stochastic	deterministic
(Locally) linear models	[7, 51, 53]	—
Gaussian Processes	[34]	[20, 21]

for using the PEGASUS algorithm for sampling trajectories from GP forward models [34] in the context of learning a blimp controller. Finally, as a fourth approach, we outline the ideas of the PILCO policy search framework [20, 21] that combines efficient deterministic approximate inference for long-term predictions with GP dynamics models for learning to control mechanical systems and robot manipulators.

3.1 Probabilistic Forward Models

To reduce the effect of model errors, probabilistic models that express uncertainty about the underlying transition dynamics are preferable to deterministic models that imply a certainty-equivalence assumption, e.g., maximum-likelihood models of the transition dynamics.

In the following, we briefly introduce two promising nonparametric probabilistic models that are frequently used for learning the forward dynamics in the context of reinforcement learning and robotics: Locally weighted Bayesian regression (LWBR), [7, 51, 53] and Gaussian processes [34, 20, 21].

3.1.1 Locally Weighted Bayesian Regression

Let us start with the linear regression model, where the transition dynamics are given as

$$\boldsymbol{x}_{t+1} = [\boldsymbol{x}_t, \boldsymbol{u}_t]^T \boldsymbol{\psi} + \boldsymbol{w}\,, \quad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_w)\,. \tag{3.3}$$

Here, $\boldsymbol{\psi}$ are the parameters of the Bayesian linear regression model, and $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_w)$ is i.i.d. Gaussian system noise. The model is linear in the unknown parameters $\boldsymbol{\psi}$ that weight the input $(\boldsymbol{x}_t, \boldsymbol{u}_t)$.

In Bayesian linear regression, we place prior distributions on the parameters $\boldsymbol{\psi}$ and on the noise variance $\boldsymbol{\Sigma}_w$. Typically, the prior distribution on $\boldsymbol{\psi}$ is Gaussian with mean $\boldsymbol{m}$ and covariance $\boldsymbol{S}$, and the prior on the diagonal entries $1/\sigma_i^2$ of $\boldsymbol{\Sigma}_w^{-1}$ is a Gamma distribution with scale and shape parameters η and $\boldsymbol{\xi}$, respectively, such that the overall model is conjugate, see Figure 3.3, where we denote the training inputs by $[\boldsymbol{X}, \boldsymbol{U}]$ and the training targets by $\boldsymbol{y}$, respectively. In the (Bayesian) linear regression model Equation (3.3), it is fairly straightforward to find maximum likelihood estimates or posterior distributions of the parameters $\boldsymbol{\psi}$. However, the model itself is not very expressive since it assumes an overall linear relationship between the inputs $(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and the successor state $\boldsymbol{x}_{t+1}$.

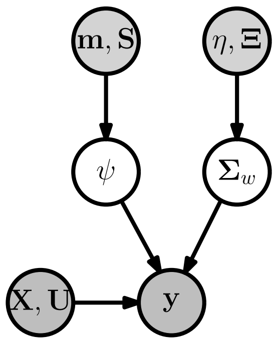


Fig. 3.3 Graphical model for Bayesian linear regression: A Gaussian prior with parameters $\boldsymbol{m}, \boldsymbol{S}$ is placed on the model parameters $\boldsymbol{\psi}$ and Gamma priors with parameters $\eta, \boldsymbol{\Xi}$ are placed on the diagonal entries of the precision matrix $\boldsymbol{\Sigma}_w^{-1}$. Training inputs and targets are denoted by $[\boldsymbol{X}, \boldsymbol{U}]$ and $\boldsymbol{y}$, respectively.

The idea of *locally weighted linear regression* (LWR) is to exploit the good properties of the linear regression model but to allow for a more general class of functions: locally linear functions. LWR finds a locally linear approximation of the underlying function [15]. For this purpose, every test input $(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is equipped with a weighting factor b_i that determines how close training point $(\boldsymbol{x}_i, \boldsymbol{u}_i)$ is to $(\boldsymbol{x}_t, \boldsymbol{u}_t)$. An example for such a weight is the Gaussian-shaped weighting $b_i = \exp(-\|(\boldsymbol{x}_i, \boldsymbol{u}_i) - (\boldsymbol{x}_t, \boldsymbol{u}_t)\|^2/\kappa^2)$. If the distance between $(\boldsymbol{x}_i, \boldsymbol{u}_i)$ and $(\boldsymbol{x}_*, \boldsymbol{u}_*)$ is much larger than κ, the corresponding weight b_i declines to 0. Since these weights have to be computed for each query point, it is insufficient to store only the parameters $\boldsymbol{\psi}$, but the entire training data set $[\boldsymbol{X}, \boldsymbol{U}]$ is required, resulting in a nonparametric approach.

As in Bayesian linear regression, we can place priors on the parameters and the noise covariance. For simplicity, let us assume a known noise covariance matrix $\boldsymbol{\Sigma}_w$ and a zero-mean prior Gaussian distribution $\mathcal{N}\big(\boldsymbol{\psi}\,|\,\mathbf{0},\boldsymbol{S}\big)$ on the parameters $\boldsymbol{\psi}$. For each query point $(\boldsymbol{x}_t,\boldsymbol{u}_t)$, a posterior distribution over the parameters $\boldsymbol{\psi}$ is computed according to Bayes' theorem as

$$p(\boldsymbol{\psi}|\boldsymbol{X},\boldsymbol{U},\boldsymbol{y}) = \frac{p(\boldsymbol{\psi})p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{U},\boldsymbol{\psi})}{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{U})} \propto p(\boldsymbol{\psi})p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{U},\boldsymbol{\psi})\,. \tag{3.4}$$

For notational convenience, we define $\tilde{\boldsymbol{X}} := [\boldsymbol{X},\boldsymbol{U}]$. The posterior mean and covariance of $\boldsymbol{\psi}$ at $(\boldsymbol{x}_t,\boldsymbol{u}_t)$ are given as

$$\mathbb{E}[\boldsymbol{\psi}|\tilde{\boldsymbol{X}},\boldsymbol{y}] = \boldsymbol{S}\tilde{\boldsymbol{X}}\boldsymbol{B}\underbrace{(\boldsymbol{B}\tilde{\boldsymbol{X}}^T\boldsymbol{S}\tilde{\boldsymbol{X}}\boldsymbol{B}+\Sigma_w)^{-1}}_{=\boldsymbol{\Omega}^{-1}}\boldsymbol{y} = \boldsymbol{S}\tilde{\boldsymbol{X}}\boldsymbol{B}\boldsymbol{\Omega}^{-1}\boldsymbol{y} \tag{3.5}$$

$$\operatorname{cov}[\boldsymbol{\psi}|\tilde{\boldsymbol{X}},\boldsymbol{y}] = \boldsymbol{S} - \boldsymbol{S}^T\tilde{\boldsymbol{X}}\boldsymbol{B}\boldsymbol{\Omega}^{-1}\boldsymbol{B}\tilde{\boldsymbol{X}}^T\boldsymbol{S}\,, \tag{3.6}$$

$$b_i = \exp(-\|(\boldsymbol{x}_i,\boldsymbol{u}_i)-(\boldsymbol{x}_t,\boldsymbol{u}_t)\|^2/\kappa^2)\,, \tag{3.7}$$

respectively, where $\boldsymbol{B} = \operatorname{diag}(b_1,\dots,b_n)$, and $\boldsymbol{y}$ are the training targets.

Predictive Distribution. The mean and covariance of the predictive distribution $p(\boldsymbol{x}_{t+1})$ for a given state-control pair $(\boldsymbol{x}_t,\boldsymbol{u}_t)$ are

$$\boldsymbol{\mu}^x_{t+1} = [\boldsymbol{x}_t,\boldsymbol{u}_t]^T\mathbb{E}[\boldsymbol{\psi}|\tilde{\boldsymbol{X}},\boldsymbol{y}] = [\boldsymbol{x}_t,\boldsymbol{u}_t]^T\boldsymbol{S}\tilde{\boldsymbol{X}}\boldsymbol{B}\boldsymbol{\Omega}^{-1}\boldsymbol{y}\,, \tag{3.8}$$

$$\boldsymbol{\Sigma}^x_{t+1} = [\boldsymbol{x}_t,\boldsymbol{u}_t]^T\operatorname{cov}[\boldsymbol{\psi}|\tilde{\boldsymbol{X}},\boldsymbol{y}][\boldsymbol{x}_t,\boldsymbol{u}_t] + \boldsymbol{\Sigma}_w\,, \tag{3.9}$$

respectively. In practice, the posterior mean and covariance over the parameters $\boldsymbol{\psi}$ can be computed more efficiently by applying matrix inversion lemmas [15] and exploiting sparsity.

Let us have a look at the predictive covariance when $[\boldsymbol{x}_t,\boldsymbol{u}_t]$ is far away from the training set $[\boldsymbol{X},\boldsymbol{U}]$: The weight matrix $\boldsymbol{B}$ is almost zero, which leads to a posterior variance over the model parameters $\boldsymbol{\psi}$ that is equal to the prior uncertainty $\boldsymbol{S}$, see Equation (3.6). Hence, the predictive variance at $[\boldsymbol{x}_t,\boldsymbol{u}_t]$ is non-zero, unlike the non-Bayesian locally weighted regression case.

3.1.2 Gaussian Process Regression

A Gaussian process is a distribution $p(f)$ over functions f. Formally, a GP is a collection of random variables $f_1, f_2, \dots$ any finite number of which is Gaussian distributed [65]. In the context of this section, a GP is placed over transition functions. Since the GP is a nonparametric model, it suffices to specify high-level assumptions, such as differentiability or periodicity, on the underlying function. These high-level properties are typically easier to specify than an explicit parametric model.

A GP is completely specified by a mean function $m(\cdot)$ and a positive semidefinite covariance function/kernel $k(\cdot,\cdot)$. Standard assumptions in GP models are a prior mean function $m \equiv 0$ and the covariance function

$$k(\tilde{\boldsymbol{x}}_p, \tilde{\boldsymbol{x}}_q) = \sigma_f^2 \exp\left(-\tfrac{1}{2}(\tilde{\boldsymbol{x}}_p - \tilde{\boldsymbol{x}}_q)^T \boldsymbol{\Lambda}^{-1} (\tilde{\boldsymbol{x}}_p - \tilde{\boldsymbol{x}}_q)\right) + \delta_{pq}\sigma_w^2 \tag{3.10}$$

with $\tilde{\boldsymbol{x}} := [\boldsymbol{x}^T \boldsymbol{u}^T]^T$. In Equation (3.10), we defined $\boldsymbol{\Lambda} := \mathrm{diag}([\ell_1^2, \ldots, \ell_D^2])$, which depends on the characteristic length-scales ℓ_i, and σ_f^2 is the prior variance of the latent function f. Given n training inputs $\tilde{\boldsymbol{X}} = [\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n]$ and corresponding training targets $\boldsymbol{y} = [y_1, \ldots, y_n]^T$, the posterior GP hyper-parameters (length-scales ℓ_i, signal variance σ_f^2, and noise variance σ_w^2) are learned using evidence maximization [43, 65].

Predictive Distribution. The posterior GP is a one-step prediction model, and the predicted successor state $\boldsymbol{x}_{t+1}$ is Gaussian distributed

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}\left(\boldsymbol{x}_{t+1} \,|\, \boldsymbol{\mu}_{t+1}^x, \boldsymbol{\Sigma}_{t+1}^x\right), \tag{3.11}$$

$$\boldsymbol{\mu}_{t+1}^x = \mathbb{E}_f[f(\boldsymbol{x}_t, \boldsymbol{u}_t)], \quad \boldsymbol{\Sigma}_{t+1}^x = \mathrm{var}_f[f(\boldsymbol{x}_t, \boldsymbol{u}_t)], \tag{3.12}$$

where the mean and variance of the GP prediction are

$$\boldsymbol{\mu}_{t+1}^x = \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{y} = \boldsymbol{k}_*^T \boldsymbol{\beta}, \tag{3.13}$$

$$\boldsymbol{\Sigma}_{t+1}^x = k_{**} - \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{k}_*, \tag{3.14}$$

respectively, with $\boldsymbol{k}_* := k(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{x}}_t)$, $k_{**} := k(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_t)$, $\boldsymbol{\beta} := \boldsymbol{K}^{-1}\boldsymbol{y}$, and where $\boldsymbol{K}$ is the kernel matrix with entries $K_{ij} = k(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)$.

Note that, far away from the training data, the predictive uncertainty in Equation (3.14) corresponds to the prior uncertainty about the function, i.e., even for deterministic systems where $\boldsymbol{\Sigma}_w = \boldsymbol{0}$, we obtain $k_{**} > 0$. Therefore, the GP is a nonparametric, non-degenerate model.

3.2 Long-Term Predictions with a Given Model

In the following, we assume that a model for the transition dynamics is known. Conditioned on this model, we distinguish between two approaches for generating long-term predictions: approaches based on Monte-Carlo sampling or deterministic approximate inference.

3.2.1 Sampling-based Trajectory Prediction: PEGASUS

PEGASUS (Policy Evaluation-of-Goodness And Search Using Scenarios) is a conceptual framework for trajectory sampling in stochastic MDPs [52]. The key idea is to transform the stochastic MDP into an augmented deterministic MDP. For this purpose, PEGASUS assumes access to a simulator with no internal random number generator. When sampling from this model, PEGASUS provides the random numbers externally in advance. In this way, PEGASUS reduces the sampling variance drastically. Therefore, sampling following the PEGASUS approach is also commonly used in model-free policy search, see Section 2.

3.2.1.1 Trajectory Sampling and Policy Evaluation

Assume that a forward model of the system at hand is given that can be used for sampling trajectories. If the state transitions are stochastic, computing the expected long-term reward

$$J_{\boldsymbol{\theta}} = \sum_{t=0}^{T} \gamma^t \mathbb{E}[r(\boldsymbol{x}_t)|\pi_{\boldsymbol{\theta}}]\,, \quad \boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)\,, \quad \gamma \in [0,1]\,, \tag{3.15}$$

will require many sample trajectories for computing the approximation $\tilde{J}$ to J, where

$$\tilde{J}_{\boldsymbol{\theta}} = \frac{1}{m}\sum_{i=1}^{m} J_{\boldsymbol{\theta}}(\boldsymbol{x}_0^{[i]}) \tag{3.16}$$

with samples $\boldsymbol{x}_0^{[i]}$ from $p(\boldsymbol{x}_0)$. Computing reliable policy gradients will require even more samples for robust derivatives. However, as the limit of an infinite number of samples, we obtain $\lim_{m\to\infty} \tilde{J}_{\boldsymbol{\theta}} = J_{\boldsymbol{\theta}}$.

For more efficient computations, PEGASUS augments the state $\boldsymbol{x}$ by an externally given sequence of random values $\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots$. To draw

Algorithm 15 PEGASUS algorithm for sampling trajectories

```
Init: $g(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$, reward $r$, random numbers $\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots$, initial state
distribution $p(\boldsymbol{x}_0)$, policy $\pi_{\boldsymbol{\theta}}$
for $i = 1, \ldots, m$ do
    $\boldsymbol{x}_0^{[i]} \sim p(\boldsymbol{x}_0)$    ▷ Sample "scenario" from initial state distribution
    for $t = 0, \ldots, T-1$ do
        $\boldsymbol{x}_{t+1}^{[i]} = g(\boldsymbol{x}_t^{[i]}, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{w}_t)$    ▷ Succ. state in augmented MDP
    end for
end for
$\tilde{J}_{\boldsymbol{\theta}} = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{T}\gamma^t r(\boldsymbol{x}_t^{[i]})$    ▷ Estimate expected long-term reward
```

a sample from $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$, PEGASUS uses a-priori given noise values $\boldsymbol{w}_t$ to compute the sampled state $\boldsymbol{x}_{t+1}$, such that $\boldsymbol{x}_{t+1} = g(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{s}_t)$. Since the sequence of random numbers is fixed, repeating the same experiment results in the identical sample trajectory.

PEGASUS can be described as generating m Monte Carlo trajectories and taking their average reward, but the randomization is determined in advance. It can be shown that solving the augmented deterministic MDP is equivalent to solving the original stochastic MDP [52]. The PEGASUS algorithm is summarized in Algorithm 15.

3.2.1.2 Practical Considerations

While sampling using the PEGASUS algorithm can be performed relatively efficiently, robust low-variance estimates of the expected long-term cost require a lot of samples. Therefore, policy search methods based on trajectory sampling are practically limited by a relatively small number of a few tens of policy parameters they can manage [50].[2] Although the policy gradients can also be computed (e.g., with finite difference approximations), the sample-based nature of PEGASUS leads to derivatives with high variance, which renders them largely useless. Thus, policy updates in the context of PEGASUS do usually not

[2] "Typically, PEGASUS policy search algorithms have been using [...] maybe on the order of ten parameters or tens of parameters; so, 30, 40 parameters, but not thousands of parameters [...]." [50]

rely on gradients [53, 7, 34]. Of course all model-free approaches from Section 2 for estimating the policy gradients can be used in conjunction with the PEGASUS idea of sampling trajectories from a given model.

An alternative to sampling trajectories are deterministic approximate inference methods for predicting trajectories, such as linearization [4], moment matching, or the unscented transformation [32].

3.2.2 Deterministic Long-Term Predictions

Instead of performing stochastic sampling, a probability distribution $p(\boldsymbol{\tau})$ over trajectories $\boldsymbol{\tau} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)$ can also be computed using deterministic approximations, such as linearization [4], sigma-point methods (e.g., the unscented transformation [32]), or moment matching. These common inference methods approximate unwieldy predictive distributions by Gaussians.

Assuming a joint Gaussian probability distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}\big([\boldsymbol{x}_t, \boldsymbol{u}_t] \,|\, \boldsymbol{\mu}_t^{xu}, \boldsymbol{\Sigma}_t^{xu}\big)$, the problem of computing the successor state distribution $p(\boldsymbol{x}_{t+1})$ corresponds to solving the integral

$$p(\boldsymbol{x}_{t+1}) = \iiint p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) p(\boldsymbol{x}_t, \boldsymbol{u}_t) \, \mathrm{d}\boldsymbol{x}_t \, \mathrm{d}\boldsymbol{u}_t \, \mathrm{d}\boldsymbol{w} \,, \tag{3.17}$$

where $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}$. If the transition function f is nonlinear, $p(\boldsymbol{x}_{t+1})$ is non-Gaussian and we have to resort to approximate inference techniques. A convenient approximation of the unwieldy predictive distribution $p(\boldsymbol{x}_{t+1})$ is the Gaussian $\mathcal{N}\big(\boldsymbol{x}_{t+1} \,|\, \boldsymbol{\mu}_{t+1}^{x}, \boldsymbol{\Sigma}_{t+1}^{x}\big)$. The mean $\boldsymbol{\mu}_{t+1}^{x}$ and covariance $\boldsymbol{\Sigma}_{t+1}^{x}$ of this predictive distribution can be computed in various ways. In the following, we outline three commonly used approaches: linearization, the unscented transformation, and moment matching.

Linearization. One way of computing $\boldsymbol{\mu}_{t+1}^{x}$ and $\boldsymbol{\Sigma}_{t+1}^{x}$ is to linearize the transition function $f \approx \boldsymbol{F}$ locally around $(\boldsymbol{\mu}_t^x, \boldsymbol{\mu}_t^u)$ and, subsequently, estimate the predictive covariance by mapping the Gaussian input distribution through the linearized system. With linearization, we obtain the predictive mean and covariance given by $\boldsymbol{\mu}_{t+1}^{x} = f(\boldsymbol{\mu}_t^{xu})$ and $\boldsymbol{\Sigma}_{t+1}^{x} = \boldsymbol{F}\boldsymbol{\Sigma}_t^{xu}\boldsymbol{F}^T + \boldsymbol{\Sigma}_w$, respectively. Figure 3.4 illustrates the idea of linearization.

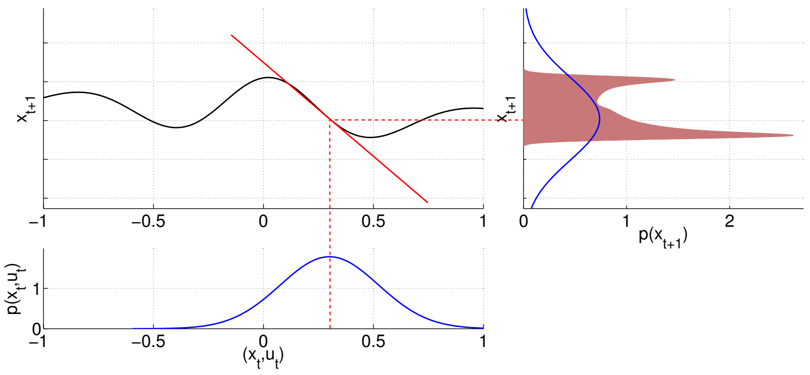


Fig. 3.4 Computing an approximate predicted distribution using linearization. A Gaussian distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ (lower-left panel) needs to be mapped through a nonlinear function (black, upper-left panel). The true predictive distribution is represented by the shaded area in the right panel. To obtain a Gaussian approximation of the unwieldy shaded distribution, the nonlinear function is linearized (red line, upper-left panel) at the mean of the input distribution. Subsequently, the Gaussian is mapped through this linear approximation and yields the blue Gaussian approximate predictive distribution $p(\boldsymbol{x}_{t+1})$ shown in the right panel.

Linearization is conceptually straightforward and computationally efficient. Note that this approach leaves the Gaussian input distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ untouched but approximates the transition function f. A potential disadvantage is that the transition function f needs to be differentiable to perform the linearization.[3] Moreover, linearization can easily underestimate predictive variances, which can cause policies to be too aggressive, causing damage on real robot systems.

Unscented Transformation. The key idea behind the unscented transformation [32] is to represent the distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ by a set of deterministically chosen sigma points $(\mathcal{X}_t^{[i]}, \mathcal{U}_t^{[i]})$. For these sigma points, the corresponding exact function values are computed. The mean $\boldsymbol{\mu}_{t+1}^x$ and covariance $\boldsymbol{\Sigma}_{t+1}^x$ of the predictive distribution $p(\boldsymbol{x}_{t+1})$ are computed from the weighted mapped sigma points. In particular,

[3] Differentiability assumptions can be problematic in robotics. For instance, contacts in locomotion and manipulation can render this assumption invalid.

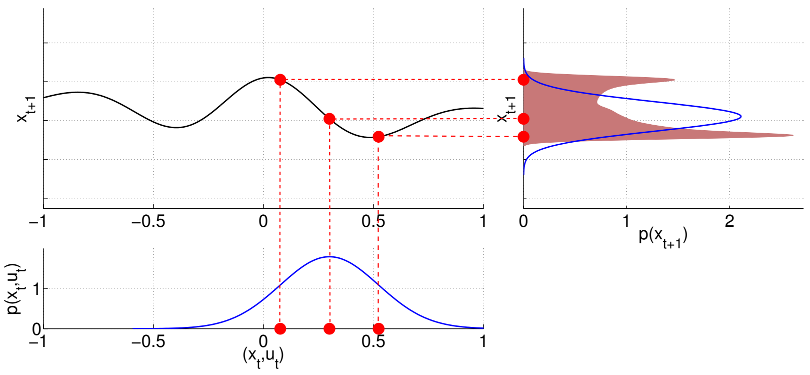


Fig. 3.5 Computing an approximate predicted distribution using the unscented transformation. A Gaussian distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ (lower-left panel) needs to be mapped through a nonlinear function (black, upper-left panel). The true predictive distribution is represented by the shaded area in the right panel. To obtain a Gaussian approximation of the unwieldy shaded distribution, the input distribution is represented by three sigma points (red dots in lower-left panel). Subsequently, the sigma points are mapped through the nonlinear function (upper-left panel) and their sample mean and covariance yield the blue Gaussian approximate predictive distribution $p(\boldsymbol{x}_{t+1})$ shown in the right panel.

we obtain

$$\boldsymbol{\mu}^x_{t+1} = \sum_{i=0}^{2d} w_m^{[i]} f(\mathcal{X}_t^{[i]}, \mathcal{U}_t^{[i]}) \,, \tag{3.18}$$

$$\boldsymbol{\Sigma}^x_{t+1} = \sum_{i=0}^{2d} w_c^{[i]} (f(\mathcal{X}_t^{[i]}, \mathcal{U}_t^{[i]}) - \boldsymbol{\mu}^x_{t+1})(f(\mathcal{X}_t^{[i]}, \mathcal{U}_t^{[i]}) - \boldsymbol{\mu}^x_{t+1})^T \,, \tag{3.19}$$

respectively, where d is the dimensionality of $(\boldsymbol{x}, \boldsymbol{u})$, $(\mathcal{X}_t^{[i]}, \mathcal{U}_t^{[i]})$ are *sigma points*, i.e., deterministically chosen samples from the joint distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$, and $w_m^{[i]}$ and $w_c^{[i]}$ are weights. For further details on the unscented transformation, we refer to [32, 82]. Figure 3.5 illustrates the idea of the unscented transformation.

Note that the unscented transformation approximates the Gaussian distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ by sigma points, which are subsequently mapped through the original transition function f. The unscented transformation does not require differentiability and is expected to yield more accurate approximations of the predictive distribution $p(\boldsymbol{x}_{t+1})$ than an explicit linearization [91].

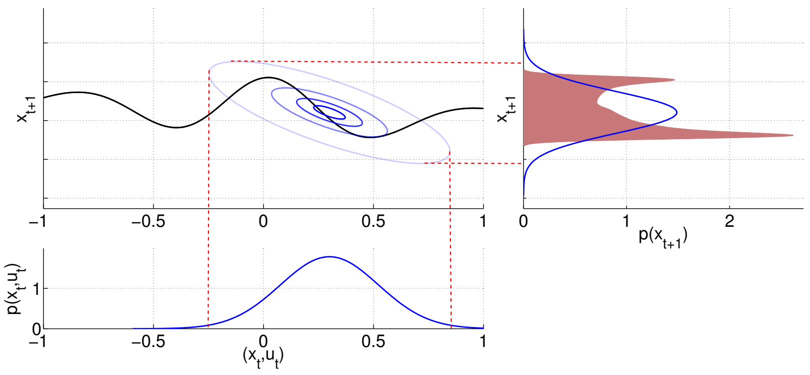


Fig. 3.6 Computing an approximate predicted distribution using moment matching. A Gaussian distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ (lower-left panel) needs to be mapped through a nonlinear function (black, upper-left panel). The true predictive distribution is represented by the shaded area in the right panel. To obtain a Gaussian approximation of the unwieldy shaded distribution, the mean and covariance of the shaded distribution are computed analytically. These first and second-order moments fully determine the blue Gaussian approximate predictive distribution $p(\boldsymbol{x}_{t+1})$ shown in the right panel. The contour lines in the upper-left panel represent the joint distribution between inputs and prediction.

Moment Matching. The idea of moment matching is to compute the predictive mean and covariance of $p(\boldsymbol{x}_{t+1})$ exactly and approximate $p(\boldsymbol{x}_{t+1})$ by a Gaussian that possesses the exact mean and covariance. Here, neither the joint distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ nor the transition function f are approximated. The moment-matching approximation is the best unimodal approximation of the predictive distribution in the sense that it minimizes the Kullback-Leibler divergence between the true predictive distribution and the unimodal approximation [12]. Figure 3.6 illustrates the idea of moment matching.

3.2.2.1 Practical Considerations

The exact moments can be computed only in special cases since the required integrals for computing the predictive mean and covariance might be intractable. Moreover, an exact moment-matching approximation is typically computationally more expensive than approximations by means of linearization or sigma points.

Unlike sampling-based approaches such as PEGASUS, deterministic approximate inference methods for long-term planning can be used to learn several thousands of policy parameters [20]. We will see examples in Section 3.4.2.1. The reason why deterministic long-term predictions can learn policies with many parameters is that gradients can be computed analytically. Therefore, these gradient estimates do not suffer from high variances, a typical problem with sampling-based estimation. Nevertheless, deterministic inference often requires more implementation effort than sampling approaches.

3.3 Policy Updates

Having introduced two major model classes and two general ways of performing long-term predictions with these models, in the following, we will discuss ways of updating the policy. We distinguish between gradient-free and gradient-based policy updates.

3.3.1 Model-based Policy Updates without Gradient Information

Gradient-free methods are probably the easiest way of updating the policy since they do not require the computation or estimation of policy gradients. By definition they also have no differentiability constraints on the policy or the transition model. Standard gradient-free optimization method are the Nelder-Mead method [47], a heuristic simplex method, or hill-climbing, a local search method that is closely related to simulated annealing [69]. Due to their simplicity and the small required computational effort, they are commonly used in the context of model-based policy search [7, 34, 51, 53], especially in combination with sampling-based trajectory generation.

A clear disadvantage of gradient-free optimization is their relatively slow convergence rate. For faster convergence, we can use gradient-based policy updates, which are introduced in the following sections.

3.3.2 Model-based Policy Updates with Gradient Information

Gradient-based policy updates are expected to yield faster convergence than gradient-free updates. We distinguish between two cases: a sample-based estimation of the policy gradients and an analytic computation of the policy gradients $\mathrm{d}J_{\boldsymbol{\theta}}(\boldsymbol{\theta})/\,\mathrm{d}\boldsymbol{\theta}$.

3.3.2.1 Sampling-based Policy Gradients

When we use sample trajectories $\boldsymbol{\tau}^{[i]}$ from the learned model to estimate the expected long-term reward $J_{\boldsymbol{\theta}}$ in Equation (3.2), we can numerically approximate the gradient $\mathrm{d}J_{\boldsymbol{\theta}}/\,\mathrm{d}\boldsymbol{\theta}$.

The easiest way of estimating gradients is to use *finite difference methods*. However, finite difference methods require $O(F)$ many evaluations of the expected long-term reward $J_{\boldsymbol{\theta}}$, where F is the number of policy parameters $\boldsymbol{\theta}$. Since each of these evaluations is based on the average of m sample roll-outs, the required number of sample trajectories quickly becomes excessive. In the model-based set-up, this is just a computational problem but not a problem of wearing the robot out since the samples are generated from the model and not the robot itself.

There are several ways of making model-based gradient estimation more efficient: First, for a more robust estimate of $J_{\boldsymbol{\theta}}$, i.e., an estimate with smaller variance, the PEGASUS approach [52] can be used. Second, for more efficient gradient estimation any of the model-free methods presented in Section 2 for gradient estimation can be used in the model-based context. The only difference is that instead of the robot, the learned model is used to generate trajectories. To the best of our knowledge, there are currently not many approaches for model-based policy search based on sampling-based gradients using the methods from Section 2.

3.3.2.2 Analytic Policy Gradients

Computing the gradients $\mathrm{d}J_{\boldsymbol{\theta}}/\,\mathrm{d}\boldsymbol{\theta}$ analytically requires the policy, the (expected) reward function, and the learned transition model to be differentiable. Despite this constraint, analytic gradient computations are a viable alternative to sampling-based gradients for two major reasons: First, they do not suffer from the sampling variance, which is especially pronounced when computing gradients. Second, the computational effort scales very favorably with the number of policy parameters, allowing for learning policies with thousands of parameters. However, due to the repeated application of the chain-rule, the computation of the gradient itself is often mathematically more involved than a sampling-based estimate.

Let us consider an example where the immediate reward r only depends on the state (generalizations to control-dependent rewards are straightforward) and the system dynamics are deterministic, such that $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) = f(\boldsymbol{x}_t, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{\theta}))$, where f is a (nonlinear) transition function, $\pi_{\boldsymbol{\theta}}$ is the (deterministic) policy, and $\boldsymbol{\theta}$ are the policy parameters. The gradient of the long-term reward $J_{\boldsymbol{\theta}} = \sum_t \gamma^t r(\boldsymbol{x}_t)$ with respect to the policy parameters is obtained by applying the chain-rule repeatedly:

$$\frac{\mathrm{d}J_{\boldsymbol{\theta}}}{\mathrm{d}\boldsymbol{\theta}} = \sum_t \gamma^t \frac{\mathrm{d}r(\boldsymbol{x}_t)}{\mathrm{d}\boldsymbol{\theta}} = \sum_t \gamma^t \frac{\partial r(\boldsymbol{x}_t)}{\partial \boldsymbol{x}_t} \frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}\boldsymbol{\theta}} \tag{3.20}$$

$$= \sum_t \gamma^t \frac{\partial r(\boldsymbol{x}_t)}{\partial \boldsymbol{x}_t} \left(\frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_{t-1}} \frac{\mathrm{d}\boldsymbol{x}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{u}_{t-1}} \frac{\mathrm{d}\boldsymbol{u}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} \right). \tag{3.21}$$

From these equations we observe that the total derivative $\mathrm{d}\boldsymbol{x}_t / \mathrm{d}\boldsymbol{\theta}$ depends on the total derivative $\mathrm{d}\boldsymbol{x}_{t-1} / \mathrm{d}\boldsymbol{\theta}$ at the previous time step. Therefore, the derivative $\mathrm{d}J_{\boldsymbol{\theta}} / \mathrm{d}\boldsymbol{\theta}$ can be computed iteratively.

Extension to Probabilistic Models and Stochastic MDPs. For the extension to derivatives in stochastic MDPs and/or probabilistic models, we have to make a few adaptations to the gradients in Equation (3.20)–(3.21): When the state $\boldsymbol{x}_t$ is represented by a probability distribution $p(\boldsymbol{x}_t)$, we have to compute the *expected* reward $\mathbb{E}[r(\boldsymbol{x}_t)] = \int r(\boldsymbol{x}_t) p(\boldsymbol{x}_t)\,\mathrm{d}\boldsymbol{x}_t$. Moreover, we need to compute the derivatives with respect to the parameters of the state distribution, assuming that $p(\boldsymbol{x}_t)$ has a parametric representation.

For example, if $p(\boldsymbol{x}_t) = \mathcal{N}\big(\boldsymbol{x}_t \mid \boldsymbol{\mu}_t^x, \boldsymbol{\Sigma}_t^x\big)$, we compute the derivatives of $\mathbb{E}[r(\boldsymbol{x}_t)]$ with respect to the mean $\boldsymbol{\mu}_t^x$ and covariance $\boldsymbol{\Sigma}_t^x$ of the state distribution and continue applying the chain-rule similarly to Equation (3.20)–(3.21): With the definition $\mathcal{E}_t := \mathbb{E}_{\boldsymbol{x}_t}[r(\boldsymbol{x}_t)]$, we obtain the gradient

$$\begin{aligned}\frac{\mathrm{d}J_{\boldsymbol{\theta}}}{\mathrm{d}\boldsymbol{\theta}} &= \sum\nolimits_t \gamma^t \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\boldsymbol{\theta}}\,,\\ \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\boldsymbol{\theta}} &= \frac{\partial \mathcal{E}_t}{\partial p(\boldsymbol{x}_t)}\frac{\mathrm{d}p(\boldsymbol{x}_t)}{\mathrm{d}\boldsymbol{\theta}} := \frac{\partial \mathcal{E}_t}{\partial \boldsymbol{\mu}_t^x}\frac{\mathrm{d}\boldsymbol{\mu}_t^x}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \mathcal{E}_t}{\partial \boldsymbol{\Sigma}_t^x}\frac{\mathrm{d}\boldsymbol{\Sigma}_t^x}{\mathrm{d}\boldsymbol{\theta}}\,, \end{aligned} \tag{3.22}$$

where we used the shorthand notation $\partial\mathcal{E}_t/\partial p(\boldsymbol{x}_t) = \{\partial\mathcal{E}_t/\partial\boldsymbol{\mu}_t^x, \partial\mathcal{E}_t/\partial\boldsymbol{\Sigma}_t^x\}$ for taking the (total) derivative of $\mathcal{E}_t$ with respect to the parameters of $p(\boldsymbol{x}_t) = \mathcal{N}\big(\boldsymbol{x}_t \,|\, \boldsymbol{\mu}_t^x, \boldsymbol{\Sigma}_t^x\big)$, i.e., the mean and covariance. The mean $\boldsymbol{\mu}_t^x$ and the covariance $\boldsymbol{\Sigma}_t^x$ are functionally dependent on the moments $\boldsymbol{\mu}_{t-1}^x$ and $\boldsymbol{\Sigma}_{t-1}^x$ of $p(\boldsymbol{x}_{t-1})$ and the controller parameters $\boldsymbol{\theta}$. By applying the chain-rule to Equation (3.22), we obtain

$$\frac{\mathrm{d}\boldsymbol{\mu}_t^x}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial\boldsymbol{\mu}_t^x}{\partial\boldsymbol{\mu}_{t-1}^x}\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}^x}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\mu}_t^x}{\partial\boldsymbol{\Sigma}_{t-1}^x}\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}^x}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\mu}_t^x}{\partial\boldsymbol{\theta}}\,, \tag{3.23}$$

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_t^x}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial\boldsymbol{\Sigma}_t^x}{\partial\boldsymbol{\mu}_{t-1}^x}\frac{\mathrm{d}\boldsymbol{\mu}_{t-1}^x}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\Sigma}_t^x}{\partial\boldsymbol{\Sigma}_{t-1}^x}\frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}^x}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial\boldsymbol{\Sigma}_t^x}{\partial\boldsymbol{\theta}}\,. \tag{3.24}$$

Note that the total derivatives $\mathrm{d}\boldsymbol{\mu}_{t-1}^x/\,\mathrm{d}\boldsymbol{\theta}$ and $\mathrm{d}\boldsymbol{\Sigma}_{t-1}^x/\,\mathrm{d}\boldsymbol{\theta}$ are known from time step $t-1$.

If all these computations can be performed in closed form, the policy gradients $\mathrm{d}\tilde{J}_{\boldsymbol{\theta}}/\,\mathrm{d}\boldsymbol{\theta}$ can be computed analytically by repeated application of the chain-rule without the need of sampling. Therefore, standard optimization techniques (e.g., BFGS or CG) can be used to learn policies with thousands of parameters [20].

3.3.3 Discussion

Using the gradients of the expected long-term reward $J_{\boldsymbol{\theta}}$ with respect to the policy parameters $\boldsymbol{\theta}$ often leads to faster learning than gradient-free policy updates. Moreover, gradient-free methods are typically limited to a few tens of policy parameters [50]. Computing the gradients can be unwieldy and requires additional computational resources. When computing gradients, exact analytic gradients are preferable over sampling-based gradients since the latter ones often suffer from large variance. These variances can even lead to slower convergence than gradient-free policy updates [7, 34]. For analytic gradients, we impose assumptions on the differentiability of the reward function r and the transition function f.[4] Moreover, for analytic gradients, we rely on deterministic approximate inference methods, e.g., moment matching or linearization, such that only an approximation $\tilde{J}_{\boldsymbol{\theta}}$ to $J_{\boldsymbol{\theta}}$ can be computed; but with the exact gradients $\mathrm{d}\tilde{J}_{\boldsymbol{\theta}}/\mathrm{d}\boldsymbol{\theta}$.

For updating the policy we recommend using gradient information to exploit better convergence properties. Ideally, the gradients are determined analytically without any approximations. Since this aim can only be achieved for linear systems, we have to resort to approximations, either by using sampling-based approaches or analytic approximate gradients. Sampling-based approaches are practically limited to fairly low-dimensional policy parameters $\boldsymbol{\theta} \in \mathbb{R}^k$, $k \leq 50$. For high-dimensional policy parameters with $k > 50$, we recommend using analytic policy gradients if they are available.

3.4 Model-based Policy Search Algorithms with Robot Applications

In this section, we briefly describe policy search methods that have been successfully applied to learning policies for robots. We distinguish between approaches that evaluate the expected long-term reward $J_{\boldsymbol{\theta}}$ using either sampling methods as described in Section 3.2.1 or deterministic approximate inference methods as described in Section 3.2.2.

3.4.1 Sampling-based Trajectory Prediction

Sampling directly from the learned simulator has been dealt with by a series of researchers for maneuvering helicopters [7, 51, 53] and for controlling blimps [34]. All approaches use the PEGASUS algorithm [52] to generate trajectories from the learned stochastic models.

Ng et al. [53, 51] learn models for hovering a helicopter based on locally weighted linear regression. To account for noise and model inaccuracies, this originally deterministic model was made stochastic by adding i.i.d. Gaussian (system) noise to the transition dynamics.

Unlike [53, 51], Bagnell and Schneider [7] explicitly describe uncer-

[4] In stochastic MDPs, this assumption is usually valid.

tainty about the learned model by means of a posterior distribution over a finite set of locally affine models. Trajectories are sampled from this mixture of models for learning the policy

Ko et al. [34] combine idealized parametric models with nonparametric Gaussian processes for modeling the dynamics of an autonomous blimp. The GPs are used to model the discrepancy between the nonlinear parametric blimp model and the data. Trajectories are sampled from this hybrid model when learning the policy. In the following, we discuss these policy search algorithms.

3.4.1.1 Locally Weighted Regression Forward Models and Sampling-based Trajectory Prediction

In [7], locally weighted Bayesian regression was used for learning forward models for hovering an autonomous helicopter. To account for model uncertainty, a posterior probability distribution over the model parameters $\boldsymbol{\psi}$ and, hence, the model itself was considered instead of a point estimate of the model parameters. Trajectories were sampled from this mixture of models for learning the policy.

Trajectories $\boldsymbol{\tau}^{[i]}$ were generated using the PEGASUS approach [52]. At each time step, a model parameter set $\boldsymbol{\psi}_i$ was sampled from the posterior distribution $p(\boldsymbol{\psi}|\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{y}, \boldsymbol{x}_t, \boldsymbol{u}_t)$. After every transition, the dynamics model was updated with the observed (simulated) transition. After each generated trajectory $\boldsymbol{\tau}^{[i]}$, the model was reset by deleting the simulated trajectory $\boldsymbol{\tau}^{[i]}$ from the model [7]. For Bayes-optimal model estimators, this procedure is equivalent to sampling the model *and* sampling a full trajectory from it. Algorithm 16 summarizes how to sample trajectories from the learned model while incorporating the posterior uncertainty about the model itself. The model uncertainty is implicitly integrated out by averaging over the expected long-term rewards for all generated trajectories $\boldsymbol{\tau}^{[i]}$.

In [7], a gradient-free optimization, the Nelder-Mead method, was used to update the policy parameters $\boldsymbol{\theta}$, which outperformed naive gradient-based optimization. The resulting approach learned a neural-network controller with ten parameters to hover a helicopter about a fixed point [7], see Figure 3.7(a). Extrapolation outside the range of

Algorithm 16 Policy evaluation and T-step predictions [7]

```
1:  Input: transition model $f$, posterior distribution over model parameters $p(\boldsymbol{\psi}|\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{y})$, policy parameters $\boldsymbol{\theta}$
2:  for $i = 1, \ldots, m$ do
3:      for $t = 0, \ldots, T-1$ do                    ▷ Sample trajectory $\boldsymbol{\tau}^{[i]}$
4:          Sample local model parameters $\boldsymbol{\psi}_i \sim p(\boldsymbol{\psi}|\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{y}, \boldsymbol{x}_t, \boldsymbol{u}_t)$
5:          Compute control $\boldsymbol{u}_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t)$
6:          Generate a sample state transition $\boldsymbol{x}_{t+1} \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\psi}_i)$
7:          Update $\boldsymbol{X}, \boldsymbol{U}$ with simulated transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1})$
8:      end for
9:      Compute $J_{\boldsymbol{\theta},i}$
10:     Reset the learned model to the original model $f$
11: end for
12: $\tilde{J}_{\boldsymbol{\theta}} = \frac{1}{m}\sum_{i=1}^{m} J_{\boldsymbol{\theta},i}$
```

(a) Helicopter hovering [7].

(b) Inverted helicopter hovering [53].

Fig. 3.7 Model-based policy search methods with stochastic inference were used for learning to hover helicopters.

collected data was discouraged by large penalties on the corresponding states. The learned controller that was based on the probability distribution over models, was substantially less oscillatory than a controller learned by using the maximum likelihood model, i.e., a point estimate of the model parameters.

Ng et al. [53, 51] learn models for helicopter hovering based on locally-weighted linear regression, see Figure 3.7(b). Unlike in [7], a point estimate of the parameters $\boldsymbol{\psi}$ in Equation (3.3) was determined, for instance by maximum likelihood or maximum-a-posteriori estima-

Fig. 3.8 Combination of a parametric prior and GPs for modeling and learning to control an autonomous blimp [34].

tion. To account for noise and model inaccuracies, this originally deterministic model was made stochastic by adding i.i.d. Gaussian (system) noise to the transition dynamics. Angular velocities expressed in helicopter coordinates were integrated and subsequently transformed into angles in world coordinates, which made the model necessarily nonlinear. With this approach, models for helicopter hovering in a standard [53] or inverse [51] pose were determined using data collected from human pilots' trajectories.

For learning a controller with these stochastic nonlinear transition dynamics, the PEGASUS [52] sampling method was used to sample trajectories from the model. With these sampled trajectories, a Monte-Carlo estimate of the expected long-term reward was computed. A greedy hill-climbing method was used to learn the parameters $\boldsymbol{\theta}$ of the policy $\pi_{\boldsymbol{\theta}}$, which was represented by a simplified neural network [51].

In the case of inverted helicopter hovering, a human pilot flipped the helicopter upside down. Then, the learned controller took over and stabilized the helicopter in the inverted position [51], an example of which is shown in Figure 3.7(b).

3.4.1.2 Gaussian Process Forward Models and Sampling-based Trajectory Prediction

In [34], GP forward models were learned to model the yaw-dynamics of an autonomous blimp, see Figure 3.8. The GP models were combined with an idealized parametric model of the blimp's dynamics, i.e., the GP modeled the discrepancy between the parametric nonlinear model and the observed data. The model was trained on blimp trajectory data generated by a human flying the blimp using a remote control.

The PEGASUS approach [52] was used to sample long-term trajectories. Each new sample was incorporated into the model by updating the kernel matrix. The controller was learned using the gradient-free Nelder-Mead [47] optimization method. The four controller parameters $\boldsymbol{\theta}$ were the drag coefficient, the right/left motor gains, and the slope of a policy-smoothing function. The learned controller was an open-loop controller, i.e., the controls were pre-computed offline and, subsequently, applied to the blimp. The controller based on the learned GP dynamics outperformed the optimal controller solely based on the underlying idealized parametric blimp dynamics model [34].

3.4.2 Deterministic Trajectory Predictions

In the following, we summarize policy search methods that perform deterministic trajectory predictions for policy evaluation.

3.4.2.1 Gaussian Process Forward Models and Deterministic Trajectory Prediction

The PILCO (probabilistic inference for learning control) policy search framework [20, 21] uses a GP forward model of the robot's dynamics to consistently account for model errors. In combination with deterministic inference by means of moment matching for predicting state trajectories $p(\boldsymbol{\tau}) = \big(p(\boldsymbol{x}_1), \dots, p(\boldsymbol{x}_T)\big)$ PILCO computes an analytic approximation $\tilde{J}$ to the expected long-term reward $J_{\boldsymbol{\theta}}$ in Equation (3.2). Moreover, the gradients $\mathrm{d}\tilde{J}/\,\mathrm{d}\boldsymbol{\theta}$ of the expected long-term reward with respect to the policy parameters are computed analytically. For policy learning, standard optimizers (e.g., BFGS or CG) can be used. The

Algorithm 17 PILCO policy search framework [20, 21]

```
Init: Sample controller parameters $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Apply random control signals and record data.
repeat
    Learn probabilistic (GP) dynamics model using all data.
    repeat
        Compute $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$ using moment matching and $\tilde{J}_{\boldsymbol{\theta}}$
        Analytically compute policy gradients $\mathrm{d}\tilde{J}_{\boldsymbol{\theta}} / \mathrm{d}\boldsymbol{\theta}$
        Update parameters $\boldsymbol{\theta}$ (line-search in BFGS or CG).
    until convergence; return $\boldsymbol{\theta}^*$
    Apply $\pi_{\boldsymbol{\theta}}{}^*$ to system (single trial/episode) and record data.
until task learned
```

PILCO algorithm is outlined in Algorithm 17. The algorithm is typically initialized uninformatively, i.e., the policy parameters are sampled randomly, and data is recorded from a short state trajectory generated by applying random actions. In the following, we briefly outline details about computing the long-term predictions, the policy gradients, and application of the PILCO framework to control and robotic systems.

Long-Term Predictions. For predicting a distribution $p(\boldsymbol{\tau}|\pi_{\boldsymbol{\theta}})$ over trajectories for a given policy, PILCO iteratively computes the state distributions $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$. For these predictions, the posterior uncertainty about the learned GP forward model is explicitly integrated out. Figure 3.9 illustrates this scenario: Let us assume, a joint Gaussian distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is given. For predicting the distribution $p(\boldsymbol{x}_{t+1})$ of the next state, the joint distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ in the lower-left panel has to be mapped through the posterior GP distribution on the latent transition function, shown in the upper-left panel. Exact inference is intractable due to the nonlinear covariance function. Extensive Monte-Carlo sampling yields a close approximation to the predictive distribution, which is represented by the shaded bimodal distribution in the right panel. Pilco computes the mean and the variance of this shaded distribution exactly and approximates the shaded distribution by a Gaussian with the correct mean and variance as shown by the

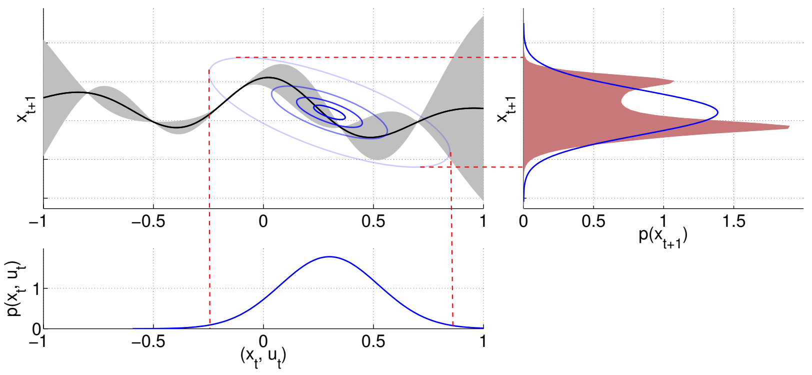


Fig. 3.9 Approximate predictions with Gaussian processes at uncertain inputs: In order to determine the predictive distribution $p(\boldsymbol{x}_{t+1})$, it is required to map the input distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ (lower-left panel) through the posterior GP distribution (upper-left panel) while explicitly averaging out the model uncertainty (shaded area). Exhaustive Monte-Carlo sampling yields the exact distribution, represented by the red shaded distribution (right panel). The deterministic moment-matching approximation computes the mean and variance of the exact predictive distribution and fits a Gaussian (blue, right panel) with the exact first two moments to it. The contour lines in the upper-left panel represent the joint distribution between inputs and prediction.

blue distribution in the upper-right panel [20, 21].

Analytic Policy Gradients. The predicted states are not point estimates but represented by Gaussian probability distributions $p(\boldsymbol{x}_t)$, $t = 1, \ldots, T$. When computing the policy gradients $\mathrm{d}J_{\boldsymbol{\theta}}/\boldsymbol{\theta}$, PILCO explicitly accounts for the probabilistic formulation by analytically computing the policy gradients for probabilistic models presented in Section 3.3.2.2.[5]

Due to the explicit incorporation of model uncertainty into long-term predictions and gradient computation, PILCO typically does not suffer severely from model errors.

Robot Applications. As shown in Figure 3.11, the PILCO algorithm achieved an unprecedented speed of learning on a standard benchmark task, the under-actuated cart-pole swing-up and balancing task, see

[5] A software package implementing the PILCO learning framework is publicly available at `http://mlg.eng.cam.ac.uk/pilco`.

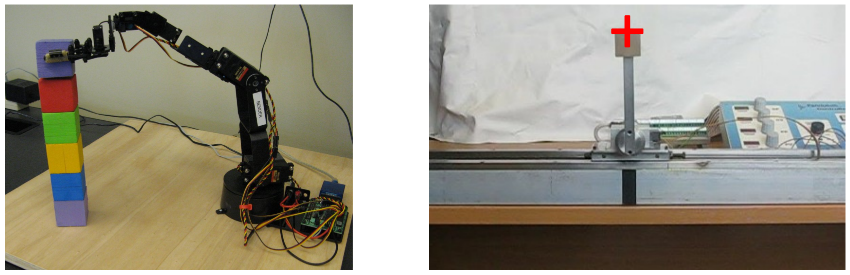


Fig. 3.10 PILCO learning successes. Left: Autonomous learning to stack a block of blocks using an off-the-shelf low-cost manipulator [21]. Right: Autonomous learning to swing up and balance a freely swinging pendulum attached to a cart [20].

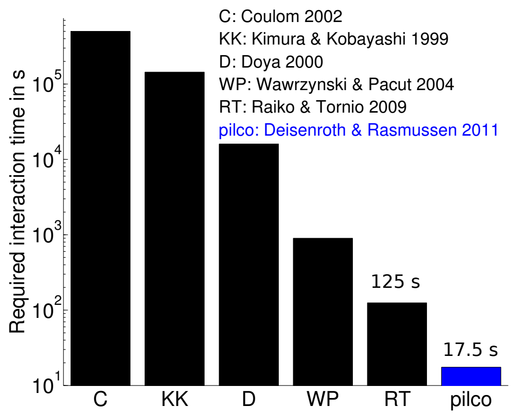


Fig. 3.11 PILCO achieves an unprecedented speed of learning on the cart-pole swing-up task. The horizontal axis gives references to RL approaches that solve the same task, the vertical axis shows the required interaction time in seconds on a logarithmic scale.

Figure 3.10. In particular, the cart-pole swing-up was learned requiring an order of magnitude less interaction time with the robot than any other RL method that also learns from scratch, i.e., without an informative initialization by demonstrations for instance. For the cart-pole swing-up problem, the learned nonlinear policy was a radial-basis function network with 50 axes-aligned Gaussian basis functions. The policy parameters $\boldsymbol{\theta}$ were the weights, the locations, and the widths of the basis functions resulting in 305 policy parameters.

PILCO was also successfully applied to efficiently learning controllers from scratch in a block-stacking task with a low-cost five degrees of

Table 3.2 Overview of model-based policy search algorithms with robotic applications.

Algorithm	Predictions	Forward Model	Policy Update	Application
[7]	sampling (PEGASUS)	LWBR	gradient free	helicopter hovering
[51, 53]	sampling (PEGASUS)	LWR+noise	gradient free	helicopter hovering
[34]	sampling (PEGASUS)	GP	gradient free	blimp control
[20, 21]	moment matching	GP	gradient based	manipulator, cart pole

freedom robot arm [21], see also Figure 3.10. The state $\boldsymbol{x}$ of the system was defined as the 3D coordinates of the block in the end-effector of the manipulator. For tracking these coordinates, an RGB-D camera was used. The learned policy $\pi_{\boldsymbol{\theta}}$ was an affine function of the state, i.e., $\boldsymbol{u} = \boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}$. Exactly following Algorithm 17, PILCO learned to stack a tower of six blocks in less than 20 trials. State-space constraints for obstacle avoidance were straightforwardly incorporated into the learning process as well [21].

3.4.3 Overview of Model-based Policy Search Algorithms

Table 3.2 summarizes the model-based policy search approaches that were presented in this section. Each algorithm is listed according to their prediction method (sampling/deterministic), their learned forward model (LWR/LWBR/GP), the policy updates (gradient free/ gradient based), and the corresponding robotic applications.

Note that in [51, 53], model errors and local minima are dealt with by injecting additional noise to the system to reduce the danger of overfitting. Generally, noise in the system (either by artificially injecting it [51, 53] or by using probabilistic models [7, 34, 20, 21]) also smoothens out the objective function and, hence, local minima.

3.5 Important Properties of Model-based Methods

In the following, we briefly discuss three important topics related to model-based policy search. In particular, we first discuss advantages and disadvantages of stochastic inference versus deterministic inference. Then, we discuss how uncertainty about the learned model itself is treated in the literature. Finally, we shed some light on the requirements when a policy is learned from scratch, i.e., learning must happen without a good initialization. The latter point is important if neither informed knowledge about the dynamics nor "good" data sets from demonstrations are available. Instead, the robot has to learn starting from, potentially sparse, data and uninformed prior knowledge.

3.5.1 Deterministic and Stochastic Long-Term Predictions

We discussed two general model-based approaches for computing distributions over trajectories and the corresponding long-term reward: Monte-Carlo sampling using the PEGASUS trick and deterministic predictions using linearization, unscented transformation, or moment matching. The advantage of stochastic sampling is that the sampler will return a correct estimate of the expected long-term reward $J_{\boldsymbol{\theta}}$ in the limit of an infinite number of sampled trajectories. Exhaustive sampling can be computationally inefficient, but it can be straightforwardly parallelized. A more significant issue with sampling, even when using the PEGASUS approach [52], is that it is only practical for several tens of policy parameters.

As an alternative to stochastic sampling, deterministic predictions only compute an exact trajectory distribution for linear-Gaussian systems. Therefore, in nonlinear systems, only an approximation to the expected long-term reward is returned. The computations required for computing predictive distributions are non-trivial and can be computationally expensive. Unlike stochastic sampling, deterministic predictions are not straightforwardly parallelizable. On the other hand, deterministic predictions have several advantages that can outweigh its disadvantages: First, despite the fact that deterministic predictions are computationally more expensive than generating a single sample transition, the requirement of many samples quickly gets computationally even more expensive. A striking advantage of deterministic predictions is that gradients with respect to the policy parameters can be computed analytically. Therefore, policy search with deterministic prediction methods can learn policies with thousands of parameters [20].

Table 3.3 summarizes the properties of deterministic and stochastic trajectory predictions. The table lists whether the expected long-term reward $J_{\boldsymbol{\theta}}$ and the corresponding gradients $\mathrm{d}J_{\boldsymbol{\theta}}/\,\mathrm{d}\boldsymbol{\theta}$ can be evaluated exactly or only approximately. For stochastic trajectory predic-

Table 3.3 Properties of deterministic and stochastic trajectory predictions in model-based policy search.

	Stochastic	Deterministic
$J_{\boldsymbol{\theta}}$	exact in the limit	approximate
$\mathrm{d}J_{\boldsymbol{\theta}}/\,\mathrm{d}\boldsymbol{\theta}$	exact in the limit	exact
Computations	simple	involved
# Policy parameters	$1 \leq \lvert\boldsymbol{\theta}\rvert \leq 50$	$1 \leq \lvert\boldsymbol{\theta}\rvert \leq$?

tions, i.e., sampling, the required computations are relatively simple whereas the computations for deterministic predictions are mathematically more involved. Finally, we give practicable bounds on the number of policy parameters that can be learned using either of the prediction methods. For stochastic trajectory generation, $J_{\boldsymbol{\theta}}$ can be evaluated exactly in the limit of infinitely many sample trajectories. The corresponding policy gradients converge even slower. In practice, where only a finite number of samples are available both $J_{\boldsymbol{\theta}}$ and $\mathrm{d}J_{\boldsymbol{\theta}}/\,\mathrm{d}\boldsymbol{\theta}$ cannot be evaluated exactly.

3.5.2 Treatment of Model Uncertainty

Expressing uncertainty about the learned model is important for model-based policy search to be robust to model errors. When predicting or generating trajectories, there are two general ways of treating model uncertainty.

In [72, 20, 21], model uncertainty is treated as *temporally uncorrelated* noise, i.e., model errors at each time step are considered independent. This approach is computationally relatively cheap and allows for the consideration of an infinite number of models during model averaging [20, 21]. Alternatively, sampling the model parameters initially and fixing the model parameters for the generated trajectory has the advantage that temporal correlation is automatically accounted for when the partially sampled trajectories are treated as training data until the model parameters are resampled [7]. Here, temporal correlation means that the state at one time step along a trajectory is correlated with the state at the previous time step. On the other hand, only a finite number of models can be sampled.

3.5.3 Extrapolation Properties of Models

In model-based policy search, it is assumed that models are known or have been trained in a pre-processing step [7, 51, 53, 34]. Here, humans were asked to maneuver the robot (e.g., a helicopter or a blimp) in order to collect data for model building. A crucial aspect of the collected data is that it covers the regions of the state space that are relevant for successfully learning the task at hand. Nevertheless, it is possible that it could be optimal (according to the reward function) to explore regions outside the training data of the current model. In this case, however, the learned model must be able to faithfully predict its confidence far away from the training data. Deterministic models (e.g., LWR or neural networks) cannot faithfully represent their confidence far away from the training data, which is why extrapolation is often discouraged by large penalty terms in the reward function [7]. Two models that possess credible error bars outside the training set are locally weighted Bayesian regression and Gaussian processes. Therefore, they can even be used for learning from scratch in a robotics context, i.e., without the need to ask a human expert to generate good data for model learning or a reasonably innate starting policy—if the robot is relatively robust to initial arbitrary exploration [20, 21].

3.5.4 Huge Data Sets

In robotics, it is not uncommon that huge data sets with millions of data points are available. For instance, recording 100 s of data at a frequency of 1 kHz leads to a data set with a million data points. For global models, such as the standard GP, these data sets sizes lead to impractical computation time. For instance, the GP would need to repeatedly store and invert a $10^6 \times 10^6$ kernel matrix during training. A common way of reducing the size of the data set is subsampling, e.g., taking only every 10th or 100th data point. This is often possible because the dynamics are sufficiently smooth, and the state of the robot does not change much in $1/1000$ s. Additionally, there are sparse approximations to the global GP [76, 63], which scale more favorably. However, even for these methods millions of data points are impractical. Therefore, local models, such as LWR or local GP models [54] should be employed if the data sets are huge. The idea of local models is to train many models with local information, and combine predictions of these models.

4

Conclusion and Discussion

In this review, we have provided an overview of successful policy search methods in the context of robot learning, where high-dimensional and continuous state-action space challenge any RL algorithm. We distinguished between model-free and model-based policy search methods.

4.1 Conclusion

Model-free policy search is very flexible as it does not make any assumptions on the underlying dynamics. Instead of learning models, data from the robot is directly used to evaluate and update the policy. When prior knowledge in the form of demonstrations or low-level policies is available, model-free policy search often yields relatively fast convergence. In Section 2, we distinguished between the used policy evaluation strategy, policy update strategy, and exploration strategy. The policy evaluation strategies can be categorized in step-based and episode-based evaluation. While step-based exploration makes more efficient use of the sampled trajectory data, algorithms using episode-based policy evaluation strategies typically learn an upper-level policy $\pi(\boldsymbol{\omega})$ which can also capture the correlation of the parameters for an efficient exploration. We presented four main policy update strategies, policy gradients, expectation-maximization, information-theoretic policy updates and policy updates based on path integrals.

Model-Free Methods. For the policy updates, we identified a main insight from information theory, i.e., the "distance" between the old trajectory distribution and the new trajectory distribution should be bounded, as an important key for a fast and stable learning process. This insight is used by the natural policy gradient algorithms [61, 62, 78] and by the REPS algorithm [57, 17]. While policy gradient methods require a user-specified learning rate which is not always easy to choose, REPS performs a weighted maximum likelihood (ML) estimate to determine the new policy, which can be obtained in closed form and does not require a learning rate.

EM-based methods such as PoWER [38] and methods based on path integrals [81] also employ a weighted maximum likelihood estimate. However, in contrast to REPS, those methods are also available in the step-based policy evaluation formulation, and, thus, might use data more efficiently. Hence, PoWER and PI^2 might show a better performance as the REPS approach in scenarios where the step-based information can be effectively exploited.

All three methods which are based on weighted ML, REPS, PoWER and PI^2 use a soft-max distribution to determine the weighting of the data-points. While in PoWER the temperature of the soft-max distribution is set by hand, PI^2 uses a heuristic which works well in practice. For the information-theoretic REPS approach, this temperature is determined by the relative entropy bound used in the algorithm and automatically recomputed for each policy update. Furthermore, episode-based REPS uses a baseline $V(\boldsymbol{s})$ to remove the state-dependent reward from the reward samples $R^{[i]}$. This baseline emerges naturally from the additional constraint to reproduce the given context distribution $p(\boldsymbol{s})$. The usage of this baseline still needs to be explored for the alternative EM and path integral approaches. Based on the beneficial properties of the information theoretic approaches, our recommendation as policy update strategy is REPS [57].

The exploration strategy creates new trajectory samples which are subsequently used to determine the policy update. Here we identified a clear trend to use an exploration strategy in parameter space which chooses the exploration only at the beginning of the episode. Furthermore, correlated exploration is preferable to uncorrelated exploration strategies as long as the number of parameters allow for an accurate estimation of the full covariance matrix used for correlated exploration strategies.

Model-Based Methods. Model-free policy search imposes only general assumptions on the entire learning process, but the number of policy parameters we can manage is limited by the number of samples that can be generated. While tens to hundred parameters are still feasible, learning several hundreds or thousands of policy parameters seems impractical due to an excessive need of real-robot experiments.

The objective of model-based policy search is to increase the data efficiency compared to model-free methods. For this purpose, an internal model of the robot is learned that, subsequently, is used for long-term predictions and policy improvement. The learned policy is, therefore, inherently limited by the quality of the model. Thus, it is crucial to account for potential model errors during policy learning by expressing uncertainty about the learned model itself. This idea has been successfully implemented by all model-based algorithms presented in Section 3. However, model learning imposes assumptions, such as differentiability, on the robot's forward dynamics, effectively reducing the generality of model-free policy search.

We distinguished between stochastic and deterministic approaches for trajectory predictions. While sampling-based inference is conceptually simple and can be easily parallelized, it is currently limited to successfully learn policies with several tens of parameters, similarly to model-free policy search. In cases with hundreds or thousands of policy parameters, we have to resort to deterministic approximate inference (e.g., linearization or moment matching), ideally in combination with an analytic computation of policy gradients. An example of a method with these characteristics is PILCO.

4.2 Current State of the Art

In the following, we qualitatively compare model-free and model-based approaches and discuss future trends in policy search for robotics.

Characteristics of Model-Free and Model-Based Policy Search Applications. Model-free policy search applications typically rely on a compact policy representation, which does not use more than 100 parameters. Typically, time-dependent representations are used, e.g., the Dynamic Movement Primitives [31, 71] approach, since such representations can encode movements for high-dimensional systems with a relatively small number of parameters. Most applications of model-free policy search rely on imitation learning to initialize the learning process.

When good models can be learned, model-based policy search is a promising alternative to model-free methods. Good models can often be learned when no abrupt changes in the dynamics occur, such as contacts in locomotion and manipulation. The presented model-based algorithms were applied to learning models for flying helicopters, blimps, and robot arms—in all cases, the underlying dynamics were relatively smooth.

Advantages of Model-Free and Model-Based Policy Search. Learning a policy is often easier than learning accurate forward models of the robot and its environment. In the literature, model-free policy search is the predominant approach in comparison to model-based methods since the difficulty of learning a model is avoided. Model-free methods do not place assumptions on the underlying process, e.g., the system dynamics do not have to be smooth. Hence, model-free policy search can also be applied to environments, which include discrete events such as hitting a table tennis ball. Episode-based policy search algorithms can also be used when the reward is not composed of the rewards of the intermediate steps.

Model-based policy search uses the learned model as a simulator of the robot. Therefore, model-based policy search is the predominant approach for fast and data-efficient learning: Once a model is learned, no interaction with the real robot is required to update the policy. Moreover, policies can be tested using the model without the risk of damaging the robot. When a model is available, the time required on the robot for running experiments is negligible.

Requirements and Limitations of Model-Free and Model-Based Policy Search. Model-free policy search methods typically require that the policy can be represented with less than 100 parameters. In addition, an initialization for the policy parameters needs to be determined, e.g., by imitation learning. In addition, model-free policy search methods are inherently local search methods and might get stuck in a local optimum.

The major advantage of model-based policy search is at the same time its major limitation: the availability of the model. Before we can exploit the model as a simulator, a sufficiently good model needs to be learned from data. By explicitly describing posterior uncertainty about the learned model itself, the effect of model errors can be reduced substantially [72, 20]. Despite the fact that non-parametric models are a very rich class of models, in some way we always need to impose smoothness assumptions to the underlying system dynamics.

4.3 Future Challenges and Research Topics.

Model-free and model-based policy search methods have, so far, been developed mostly in isolation. However, the combination of model-free policy search with learned models seems to be a promising approach, a recent example is given in [41]. For example, most model-based approaches greedily exploit the learned model by using gradient-based approaches. Model-free policy update strategies could be used to avoid a greedy optimization, and, hence, the additional exploration might in the end improve the quality of the learned models. Another promising approach is to smoothly switch from model-based policy search in the initial learning phase to model-free policy search when sufficiently much data is available for model-free policy updates. In such case, the model could guide the initial exploration into relevant areas of the parameter space. For fine-tuning the policy, real trajectory samples are used, and, hence, the policy update is not affected from model errors.

We believe that one of the most important challenges in robot learning is to incorporate structured and hierarchical learning methods into policy search. Many motor tasks are structured. For example, many tasks can be decomposed into elemental movements, also called movement primitives or options. Such a decomposition suggests a modular control approach wherein options can be adapted to the current context, activated simultaneously and sequentially in time. While there have been first approaches in model-free policy search to adapt options to the current context as well as to select options with a gating network, hierarchical approaches have so far not been explored for model-based reinforcement learning. Extending model-free methods with learned models seems promising in the field of hierarchical policy search. In general, we believe that the use of hierarchical robot control policies used in robot learning is largely unexplored and will become a new important subfield in robot learning.

In model-free policy search, we think it is important to generate a unified framework for imitation learning and reinforcement learning, such that data from both approaches can be used within one algorithm. Further goals in model-free policy search include developing a principled way to combine the advantages of step-based policy search algorithms and episode-based algorithms. This means to combine the effective use of sampled trajectory data from step-based methods with more sophisticated exploration strategies and the extensibility to hierarchical policies. Furthermore, the advantages of step-based and episode-based algorithms need to be explored in more detail, e.g., for which policy representation which type of algorithm is more useful. In addition, we believe that a principled treatment of exploration as individual objective for policy search algorithms is a promising approach to achieve a more stable learning process and avoid heuristics such as adding additional noise terms to sustain exploration.

For model-based approaches, the main challenges are choosing an appropriate model class and performing long-term predictions with this model. Non-parametric methods, such as Gaussian processes or LWBR already provide the necessary flexibility but might be difficult to scale to high-dimensional systems. In order to model discontinuities, the use of hierarchical forward models is promising approach. The hierarchy can either be defined by the user or directly inferred from the data [27]. Once an appropriate model class has been chosen, the model can be used for long-term predictions and policy evaluation. For nonlinear model classes, approximate inference is required. Depending on the chosen model class and the number of policy parameters, we can choose between stochastic approximate inference, i.e., sampling, or deterministic approximate inference, e.g., moment matching. Since the policy representation, the learned model, and the policy update strategy are inherently connected, all these components need to fit well together in order to make the overall learning process successful.

A key challenge in learning for robots is to deal with sensory information: First, sensor data is typically noisy. Especially for model learning purposes, noisy data is challenging since not only the measurements but also the training inputs are noisy. This fact is often tacitly ignored in practice. Second, sensor data, such as images, can be high dimensional. Dimensionality reduction and feature learning can be used for a lower-dimensional compact representation of the data. Therefore, robot learning with noisy and high-dimensional data can also be phrased in the context of learning and solving partially observable Markov decision processes (MDPs) with continuous state and control spaces.

Finally, the field of robot learning needs to move to more complex applications. So far, many applications included single-stroke tasks such as hitting a baseball or catching a ball with a cup. The next step is to integrate robot learning into large-scale tasks which require the execution of a multitude of single movements. Challenging examples are given by dexterous manipulation, legged locomotion on uneven terrain, playing a full game of table tennis against a human champion, or learning to play soccer with humanoid robots. For these complex tasks, it will be necessary to exploit their modularity to simplify the learning problem. Ideally, we would automatically create re-usable submodules for policies and models that can be combined to complex policies and models.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement #270327.

References

[1] P. Abbeel, M. Quigley, and A. Y. Ng. Using Inaccurate Models in Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 1–8, Pittsburgh, PA, USA, June 2006.

[2] E. W. Aboaf, S. M. Drucker, and C. G. Atkeson. Task-Level Robot Learning: Juggling a Tennis Ball More Accurately. In *Proceedings of the International Conference on Robotics and Automation*, 1989.

[3] S. Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10:251–276, February 1998.

[4] B. D. O. Anderson and J. B. Moore. *Optimal Filtering.* Dover Publications, Mineola, NY, USA, 2005.

[5] K. J. Aström and B. Wittenmark. *Adaptive Control.* Dover Publications, 2008.

[6] C. G. Atkeson and J. C. Santamaría. A Comparison of Direct and Model-Based Reinforcement Learning. In *Proceedings of the International Conference on Robotics and Automation*, 1997.

[7] J. A. Bagnell and J. G. Schneider. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *Proceed-*

ings of the International Conference on Robotics and Automation, pages 1615–1620. IEEE Press, 2001.

[8] J. A. Bagnell and J. G. Schneider. Covariant Policy Search. In *Proceedings of the International Joint Conference on Artifical Intelligence*, August 2003.

[9] J. Baxter and P. Bartlett. Direct Gradient-Based Reinforcement Learning: I. Gradient Estimation Algorithms. Technical report, 1999.

[10] J. Baxter and P. L. Bartlett. Infinite-Horizon Policy-Gradient Estimation. *Journal of Artificial Intelligence Research*, 2001.

[11] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1 of *Optimization and Computation Series*. Athena Scientific, Belmont, MA, USA, 3rd edition, 2005.

[12] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag, 2006.

[13] J. A. Boyan. Least-Squares Temporal Difference Learning. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56, 1999.

[14] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[15] W. S. Cleveland and S. J. Devlin. Locally-Weighted Regression: An Approach to Regression Analysis by Local Fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.

[16] R. Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[17] C. Daniel, G. Neumann, and J. Peters. Hierarchical Relative Entropy Policy Search. In N. Lawrence and M. Girolami, editors, *Proceedings of the International Conference of Artificial Intelligence and Statistics*, pages 273–281, 2012.

[18] C. Daniel, G. Neumann, and J. Peters. Learning Concurrent Motor Skills in Versatile Solution Spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[19] P. Dayan and G. E. Hinton. Using Expectation-Maximization for Reinforcement Learning. *Neural Computation*, 9(2):271–278, 1997.

[20] M. P. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based

and Data-Efficient Approach to Policy Search. In *Proceedings of the International Conference on Machine Learning*, pages 465–472, New York, NY, USA, June 2011. ACM.

[21] M. P. Deisenroth, C. E. Rasmussen, and D. Fox. Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. In *Proceedings of the International Conference on Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[22] M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian Process Dynamic Programming. *Neurocomputing*, 72(7–9):1508–1524, March 2009.

[23] K. Doya. Reinforcement Learning in Continuous Time and Space. *Neural Computation*, 12(1):219–245, January 2000.

[24] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng. Learning CPG-based Biped Locomotion with a Policy Gradient Method: Application to a Humanoid Robot. *International Journal of Robotics Research*, 2008.

[25] S. Fabri and V. Kadirkamanathan. Dual Adaptive Control of Nonlinear Stochastic Systems using Neural Networks. *Automatica*, 34(2):245–253, 1998.

[26] A. A. Fel'dbaum. Dual Control Theory, Parts I and II. *Automation and Remote Control*, 21(11):874–880, 1961.

[27] E. B. Fox and D. B. Dunson. Multiresolution Gaussian Processes. In *Advances in Neural Information Processing Systems*. The MIT Press, 2012.

[28] N. Hansen, S. Muller, and P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[29] V. Heidrich-Meisner and C. Igel. Hoeffding and Bernstein Races for Selecting Policies in Evolutionary Direct Policy Search. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 401–408. ACM, 2009.

[30] V. Heidrich-Meisner and C. Igel. Neuroevolution Strategies for Episodic Reinforcement Learning. *Journal of Algorithms*, 64(4):152–168, oct 2009.

[31] A. J. Ijspeert and S. Schaal. Learning Attractor Landscapes for

Learning Motor Primitives. In *Advances in Neural Information Processing Systems*, pages 1523–1530. MIT Press, Cambridge, MA, 2003.

[32] S. J. Julier and J. K. Uhlmann. Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, 92(3):401–422, March 2004.

[33] H. Kimura and S. Kobayashi. Efficient Non-Linear Control by Combining Q-learning with Local Linear Controllers. In *Proceedings of the 16th International Conference on Machine Learning*, pages 210–219, 1999.

[34] J. Ko, D. J. Klein, D. Fox, and D. Haehnel. Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp. In *Proceedings of the International Conference on Robotics and Automation*, pages 742–747, 2007.

[35] J. Kober, B. J. Mohler, and J. Peters. Learning Perceptual Coupling for Motor Primitives. In *Intelligent Robots and Systems*, pages 834–839, 2008.

[36] J. Kober, K. Mülling, O. Kroemer, C. H. Lampert, B. Schölkopf, and J. Peters. Movement Templates for Learning of Hitting and Batting. In *International Conference on Robotics and Automation*, pages 853–858, 2010.

[37] J. Kober, E. Oztop, and J. Peters. Reinforcement Learning to adjust Robot Movements to New Situations. In *Proceedings of the 2010 Robotics: Science and Systems Conference*, 2010.

[38] J. Kober and J. Peters. Policy Search for Motor Primitives in Robotics. *Machine Learning*, pages 1–33, 2010.

[39] N. Kohl and P. Stone. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In *Proceedings of the International Conference on Robotics and Automation*, 2003.

[40] P. Kormushev, S. Calinon, and D. G. Caldwell. Robot Motor Skill Coordination with EM-based Reinforcement Learning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[41] A. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann. Data-Efficient Generalization of Robot Skills with Contextual Policy Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2013.

[42] M. G. Lagoudakis and R. Parr. Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 4:1107–1149, December 2003.

[43] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 2003.

[44] D. C. McFarlane and K. Glover. *Lecture Notes in Control and Information Sciences*, volume 138, chapter Robust Controller Design using Normalised Coprime Factor Plant Descriptions. Springer-Verlag, 1989.

[45] J. Morimoto and C. G. Atkeson. Minimax Differential Dynamic Programming: An Application to Robust Biped Walking. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*. The MIT Press, 2003.

[46] R. Neal and G. E. Hinton. A View Of The EM Algorithm That Justifies Incremental, Sparse, And Other Variants. In *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998.

[47] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7:308–313, 1965.

[48] G. Neumann. Variational Inference for Policy Search in Changing Situations. In *Proceedings of the 28th International Conference on Machine Learning*, pages 817–824, New York, NY, USA, June 2011. ACM.

[49] G. Neumann and J. Peters. Fitted Q-Iteration by Advantage Weighted Regression. In *Neural Information Processing Systems*. MA: MIT Press, 2009.

[50] A. Y. Ng. Stanford Engineering Everywhere CS229—Machine Learning, Lecture 20, 2008. `http://see.stanford.edu/materials/aimlcs229/transcripts/MachineLearning-Lecture20.html`.

[51] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous Inverted Helicopter Flight via Reinforcement Learning. In M. H. Ang Jr. and O. Khatib, editors, *International Symposium on Experimental Robotics*, vol-

ume 21 of *Springer Tracts in Advanced Robotics*, pages 363–372. Springer, 2004.

[52] A. Y. Ng and M. Jordan. PEGASUS: A Policy Search Method for Large MDPs and POMDPs. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

[53] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Autonomous Helicopter Flight via Reinforcement Learning. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, Cambridge, MA, USA, 2004. The MIT Press.

[54] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model Learning with Local Gaussian Process Regression. *Advanced Robotics*, 23(15):2015–2034, 2009.

[55] B. Øksendal. *Stochastic Differential Equations: An Introduction with Applications (Universitext)*. Springer, 6th edition, Sept. 2010.

[56] J. Peters, M. Mistry, F. E. Udwadia, J. Nakanishi, and S. Schaal. A Unifying Methodology for Robot Control with Redundant DOFs. *Autonomous Robots*, (1):1–12, 2008.

[57] J. Peters, K. Mülling, and Y. Altun. Relative Entropy Policy Search. In *Proceedings of the 24th National Conference on Artificial Intelligence*. AAAI Press, 2010.

[58] J. Peters and S. Schaal. Policy Gradient Methods for Robotics. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robotics Systems*, pages 2219–2225, Beijing, China, 2006.

[59] J. Peters and S. Schaal. Applying the Episodic Natural Actor-Critic Architecture to Motor Primitive Learning. In *Proceedings of the European Symposium on Artificial Neural Networks*, 2007.

[60] J. Peters and S. Schaal. Natural Actor-Critic. *Neurocomputation*, 71(7-9):1180–1190, 2008.

[61] J. Peters and S. Schaal. Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks*, (4):682–97, 2008.

[62] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement Learning for Humanoid Robotics. In *IEEE-RAS International Conference on Humanoid Robots*. IEEE, September 2003.

[63] J. Quiñonero-Candela and C. E. Rasmussen. A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of*

Machine Learning Research, 6(2):1939–1960, 2005.

[64] T. Raiko and M. Tornio. Variational Bayesian Learning of Nonlinear Hidden State-Space Models for Model Predictive Control. *Neurocomputing*, 72(16–18):3702–3712, 2009.

[65] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA, 2006.

[66] L. Rozo, S. Calinon, D. G. Caldwell, P. Jimenez, and C. Torras. Learning collaborative impedance-based robot behaviors. In *AAAI Conference on Artificial Intelligence*, Bellevue, Washington, USA, 2013.

[67] T. Rückstieß, M. Felder, and J. Schmidhuber. State-Dependent Exploration for Policy Gradient Methods. In *European Conference on Machine Learning*, pages 234–249, 2008.

[68] T. Rückstieß, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber. Exploring Parameter Space in Reinforcement Learning. *Paladyn*, 1(1):14–24, March 2010.

[69] S. J. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[70] S. Schaal and C. G. Atkeson. Constructive Incremental Learning from only Local Information. *Neural Computation*, 10(8):2047–2084, 1998.

[71] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning Movement Primitives. In *International Symposium on Robotics Research*, pages 561–572, 2003.

[72] J. G. Schneider. Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning. In *Advances in Neural Information Processing Systems*. Morgan Kaufman Publishers, 1997.

[73] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Policy Gradients with Parameter-based Exploration for Control. In *Proceedings of the International Conference on Artificial Neural Networks*, 2008.

[74] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Parameter-Exploring Policy Gradients. *Neural Networks*, 23(4):551–559, 2010.

[75] C. Shu, H. Ding, and N. Zhao. Numerical Comparison of Least Square-Based Finite-Difference (LSFD) and Radial Basis Function-Based Finite-Difference (RBFFD) Methods. *Computers & Mathematics with Applications*, 51(8):1297–1310, April 2006.

[76] E. Snelson and Z. Ghahramani. Sparse Gaussian Processes using Pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264. The MIT Press, Cambridge, MA, USA, 2006.

[77] F. Stulp and O. Sigaud. Path Integral Policy Improvement with Covariance Matrix Adaptation. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[78] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber. Efficient Natural Evolution Strategies. In *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation*, pages 539–546, New York, NY, USA, 2009. ACM.

[79] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Neural Information Processing Systems*, 1999.

[80] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. The MIT Press, Boston, MA, 1998.

[81] E. Theodorou, J. Buchli, and S. Schaal. A Generalized Path Integral Control Approach to Reinforcement Learning. *Journal of Machine Learning Research*, (11):3137–3181, 2010.

[82] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, Cambridge, MA, USA, 2005.

[83] E. Todorov. Optimal Control Theory. *Bayesian Brain*, 2006.

[84] M. Toussaint. Robot Trajectory Optimization using Approximate Inference. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

[85] N. Vlassis and M. Toussaint. Model-Free Reinforcement Learning as Mixture Learning. In *International Conference on Machine Learning*, page 136, 2009.

[86] N. Vlassis, M. Toussaint, G. Kontes, and S. Piperidis. Learning Model-Free Robot Control by a Monte Carlo EM Algorithm. *Autonomous Robots*, 27(2):123–130, 2009.

[87] P. Wawrzynski and A. Pacut. Model-Free Off-Policy Reinforcement Learning in Continuous Environment. In *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, pages 1091–1096, 2004.

[88] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural Evolution Strategies. In *IEEE Congress on Evolutionary Computation*, pages 3381–3387, 2008.

[89] R. J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

[90] B. Wittenmark. Adaptive Dual Control Methods: An Overview. In *In Proceedings of the 5th IFAC Symposium on Adaptive Systems in Control and Signal Processing*, pages 67–72, 1995.

[91] K. Xiong, H.-Y. Zhang, and C. W. Chan. Performance Evaluation of UKF-based Nonlinear Filtering. *Automatica*, 42:261–270, 2006.

A Gradients of Frequently Used Policies

All used policies use Gaussian distributions to generate exploration. Here we state the most frequently used gradients for Gaussian policies w.r.t the mean and the covariance matrix of the Gaussian. The gradients are always stated for policies in action space. However, the policies which are defined in parameter space have of course the same gradient.

The log-likelihood of a Gaussian policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given by

$$\log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) = -\frac{d}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}\left(\boldsymbol{u}-\boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1}\left(\boldsymbol{u}-\boldsymbol{\mu}\right).$$

Constant Mean. If the policy is given as $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{\mu}$ is part of $\boldsymbol{\theta}$, then

$$\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) = (\boldsymbol{u}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}.$$

The gradient simplifies if $\boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\sigma}^2)$,

$$\nabla_{\mu_d} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) = (u_d - \mu_d)/\sigma_d^2,$$

for the d-th dimension of $\boldsymbol{\mu}$.

Linear Mean. If the policy is given as $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{\phi}_t(\boldsymbol{x})^T\boldsymbol{M}, \boldsymbol{\Sigma})$ and $\boldsymbol{M}$ is part of $\boldsymbol{\theta}$, then

$$\nabla_{\boldsymbol{M}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) = (\boldsymbol{u} - \boldsymbol{\phi}_t(\boldsymbol{x})^T\boldsymbol{M})^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\phi}_t(\boldsymbol{x}),$$

or for a diagonal covariance matrix

$$\nabla_{\boldsymbol{m}_d} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) = (u_d - \boldsymbol{\phi}_t(\boldsymbol{x})^T\boldsymbol{m}_d)\boldsymbol{\phi}_t(\boldsymbol{x})/\sigma_d^2,$$

where $\boldsymbol{m}_d$ corresponds to the d-th column of $\boldsymbol{M}$.

Diagonal Covariance Matrix. For a diagonal covariance matrix $\boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\sigma}^2)$ the derivative is given by

$$\nabla_{\sigma_d} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) = -\frac{1}{\sigma_d^2} + \frac{(u_d-\mu_d)^2}{\sigma_d^3}$$

Full Covariance Matrix. For representing the full covariance matrix, typically the Cholesky decomposition of the covariance matrix $\boldsymbol{\Sigma} = \boldsymbol{A}^T\boldsymbol{A}$, where $\boldsymbol{A}$ is an upper-triangular matrix, is used as parametrization [78]. The parametrization with the Cholesky decomposition exploits the symmetry of the covariance matrix and enforces that $\boldsymbol{\Sigma}$ is positive definite. The gradient of $\log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x})$ w.r.t $\boldsymbol{A}$ is given by

$$\begin{aligned}\partial_{a_{i,j}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{u}|\boldsymbol{x}) &= -\frac{1}{2}\partial_{a_{i,j}} \log |\boldsymbol{A}^T\boldsymbol{A}| - \\ &\quad \frac{1}{2}\partial_{a_{i,j}} \left(\boldsymbol{A}^{-T}(\boldsymbol{u}-\boldsymbol{\mu})\right)^T \left(\boldsymbol{A}^{-T}(\boldsymbol{u}-\boldsymbol{\mu})\right) \\ &= a_{i,j}^{-1}\delta_{i,j} + s_{i,j},\end{aligned}$$

where $\delta_{i,j}$ is the dirac-delta function which is one if $i = j$ and zero elsewhere and $s_{i,j}$ is the (i,j)th element of the matrix $\boldsymbol{S}$,

$$\boldsymbol{S} = (\boldsymbol{u}-\boldsymbol{\mu})(\boldsymbol{u}-\boldsymbol{\mu})^T \boldsymbol{A}^{-1}\boldsymbol{A}^{-T}\boldsymbol{A}^{-1}.$$

B Weighted ML Estimates of Frequently Used Policies

We assume a data-set $\mathcal{D}$ given in the following form

$$\mathcal{D} = \left\{\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]}, d^{[i]}\right\}_{i=1\ldots N},$$

where $d^{[i]}$ denotes the weighting of the ith sample. In this section we will state the solution of weighted ML estimation, i.e.,

$$\boldsymbol{\theta}^* = \operatorname{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log \pi_{\boldsymbol{\theta}}\left(\boldsymbol{u}^{[i]}\middle|\boldsymbol{x}^{[i]}\right),$$

for the most frequently used policies.

For the episode-based formulation of Policy-Search, the states $\boldsymbol{x}^{[i]}$ are exchanged with the contexts $\boldsymbol{s}^{[i]}$ and the actions $\boldsymbol{u}^{[i]}$ are exchanged by the parameters $\boldsymbol{\theta}^{[i]}$ of the lower level controller.

Gaussian Policy, Constant Mean. Consider a policy which is given by $\pi(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, i.e., we do not have a state or a context. Such a policy is for example useful to model the upper-level policy without contexts. The weighted ML-solution for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ is given by

$$\boldsymbol{\mu} = \frac{\sum_{i=1}^{N} d^{[i]} \boldsymbol{u}^{[\boldsymbol{i}]}}{\sum_{i=1}^{N} d^{[i]}}, \quad \boldsymbol{\Sigma} = \frac{\sum_{i=1}^{N} d^{[i]} \left(\boldsymbol{u}^{[i]} - \boldsymbol{\mu}\right) \left(\boldsymbol{u}^{[i]} - \boldsymbol{\mu}\right)^T}{Z}, \tag{4.1}$$

where

$$Z = \frac{\left(\sum_{i=1}^{N} d^{[i]}\right)^2 - \sum_{i=1}^{N} \left(d^{[i]}\right)^2}{\sum_{i=1}^{N} d^{[i]}}$$

is used to obtain an unbiased estimate of the covariance. The elements σ of a diagonal covariance matrix $\boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\sigma})$ can be obtained by

$$\sigma_h = \frac{\sum_{i=1}^{N} d^{[i]} \left(\boldsymbol{u}_h^{[i]} - \mu_h\right)^2}{Z}. \tag{4.2}$$

Gaussian Policy, Linear Mean. The policy is given by $\pi(\boldsymbol{u}|x) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{W}^T\boldsymbol{\phi}(x), \boldsymbol{\Sigma})$. The weighted ML-solution for $\boldsymbol{W}$ is determined by the weighted pseudo-inverse

$$\boldsymbol{W}_{\text{new}} = (\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{U}, , \tag{4.3}$$

where $\boldsymbol{\Phi} = \left[\boldsymbol{\phi}^{[1]}, \ldots, \boldsymbol{\phi}^{[N]}\right]$ contains the feature vectors for all samples and $\boldsymbol{D}$ is the diagonal weighting matrix containing the weightings $d_t^{[i]}$. The covariance matrix $\boldsymbol{\Sigma}$ is obtained by

$$\Sigma = \frac{\sum_{i=1}^{N} d^{[i]} \left(\boldsymbol{u}^{[i]} - \boldsymbol{W}^T\boldsymbol{\phi}(x^{[i]})\right) \left(\boldsymbol{u}^{[i]} - \boldsymbol{W}^T\boldsymbol{\phi}(x^{[i]})\right)^T}{Z}, \tag{4.4}$$

where Z is defined as in Eq.(4.1).

Gaussian Policy, Linear Mean, State-Dependent Variance The policy is given by $\pi(u|x) = \mathcal{N}(u|\boldsymbol{w}^T\boldsymbol{\phi}(x), \boldsymbol{\phi}(x)^T\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\phi}(x))$. Here, we consider only the scalar case as the multi-dimensional distribution case is more complicated and only rarely used. The weighted ML-solution for $\boldsymbol{w}$ is determined by the weighted pseudo-inverse where the weights $\tilde{d}^{[i]}$ are given as the product of the state-dependent precision $\boldsymbol{\phi}(x^{[i]})^T\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\phi}(x^{[i]})$ and the actual weighting $d^{[i]}$. The solution for $\boldsymbol{w}$ is equivalent to Equation (4.3) where $\boldsymbol{D}$ is set to the new weightings $\tilde{d}^{[i]}$.

C Derivations of the Dual Functions for REPS

In this section we will briefly discuss constraint optimization, Lagrangian multipliers and dual-functions in general. Subsequently, we will derive the dual-functions for the different REPS formulations.

Lagrangian Function. Consider the following general constraint optimization problem with equality and inequality constraints

$$\begin{aligned} \max_{\boldsymbol{y}} \quad & f(\boldsymbol{y}) \\ \text{s.t:} \quad & \boldsymbol{a}(\boldsymbol{y}) = \boldsymbol{0} \\ & \boldsymbol{b}(\boldsymbol{y}) \leq \boldsymbol{0} \end{aligned} \tag{4.5}$$

Such optimization problem can be solved by finding the saddle-points of the Lagrangian

$$L = f(\boldsymbol{y}) + \boldsymbol{\lambda}_1^T \boldsymbol{a}(\boldsymbol{y}) + \boldsymbol{\lambda}_2^T \boldsymbol{b}(\boldsymbol{y}). \tag{4.6}$$

The optimization problem has a local maximum if the direction of the gradient $\partial_{\boldsymbol{y}} f(\boldsymbol{y})$ is aligned with the normal of the constraints $\boldsymbol{\lambda}_1^T \partial_{\boldsymbol{y}} a(\boldsymbol{y})$ and $\boldsymbol{\lambda}_2^T \partial_{\boldsymbol{y}} b(\boldsymbol{y})$. Such a point can be found by differentiating the Lagrangian w.r.t $\boldsymbol{y}$ and setting it to zero.

$$\partial_{\boldsymbol{y}} f(\boldsymbol{y}) = \boldsymbol{\lambda}_1^T \partial_{\boldsymbol{y}} \boldsymbol{a}(\boldsymbol{y}) + \boldsymbol{\lambda}_2^T \partial_{\boldsymbol{y}} \boldsymbol{b}(\boldsymbol{y}). \tag{4.7}$$

Dual Function. Optimizing the dual function of an optimization problem is, under certain conditions equivalent to solving the original optimization problem [14]. However, the dual function is often easier to optimize. The dual function is obtained by finding $\boldsymbol{y} = \boldsymbol{c}(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2)$ which satisfies the saddle-point condition given in Equation (4.7). This solution is in turn set back into the Lagrangian, which results in the dual-function $g(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2)$.

If the original optimization problem is maximized, the dual function needs to be minimized [14]. The dual function only depends on the Lagrangian multipliers and is therefore often easier to optimize. Each inequality constraint used in the original optimization problem introduces an inequality constraint for the Lagrangian multipliers. Hence, the original optimization problem can also be solved by solving the following program

$$\begin{aligned} \min_{\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2} \quad & g(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) \\ \text{s.t:} \quad & \boldsymbol{\lambda}_2 \geq \mathbf{0} \end{aligned} \tag{4.8}$$

The solution for $\boldsymbol{y}$ can subsequently be found by setting the Lagrangian parameters back into $\boldsymbol{c}(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2)$.

Step-Based REPS. We denote $p(\boldsymbol{x}, \boldsymbol{u}) = \mu^{\pi}(\boldsymbol{x})\pi(\boldsymbol{u}|\boldsymbol{x})$ and $p(\boldsymbol{x}) = \sum_{\boldsymbol{u}} p(\boldsymbol{x}, \boldsymbol{u})$ for brevity of the derivations. To simplify the derivations, we will also write all integrals as sums. However, the derivations also hold for the formulation with the integrals. The Lagrangian for the program in Equation (2.79) with state features $\boldsymbol{\varphi}(\boldsymbol{x})$ is given by

$$\begin{aligned} L = & \left(\sum_{\boldsymbol{x},\boldsymbol{u}} p(\boldsymbol{x}, \boldsymbol{u}) r(\boldsymbol{x}, \boldsymbol{u})\right) + \eta \left(\epsilon - \sum_{\boldsymbol{x},\boldsymbol{u}} p(\boldsymbol{x}, \boldsymbol{u}) \log \frac{p(\boldsymbol{x}, \boldsymbol{u})}{q(\boldsymbol{x}, \boldsymbol{u})}\right) \\ & + \boldsymbol{v}^T \sum_{\boldsymbol{x}'} \boldsymbol{\varphi}(\boldsymbol{x}') \left(\sum_{\boldsymbol{x},\boldsymbol{u}} p(\boldsymbol{x}, \boldsymbol{u}) p(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{u}) - \sum_{\boldsymbol{u}'} p(\boldsymbol{x}', \boldsymbol{u}')\right) \\ & + \lambda \left(1 - \sum_{\boldsymbol{x},\boldsymbol{u}} p(\boldsymbol{x}, \boldsymbol{u})\right), \end{aligned} \tag{4.9}$$

where η, $\boldsymbol{v}$ and λ denote the Lagrangian multipliers. Rearranging terms results in

$$\begin{aligned} L = \sum_{\boldsymbol{x},\boldsymbol{u}} p(\boldsymbol{x}, \boldsymbol{u}) & \left(r(\boldsymbol{x}, \boldsymbol{u}) - \eta \log \frac{p(\boldsymbol{x}, \boldsymbol{u})}{q(\boldsymbol{x}, \boldsymbol{u})} - \lambda - \boldsymbol{v}^T \boldsymbol{\varphi}(\boldsymbol{x}) \right. \\ & \left. + \boldsymbol{v}^T \sum_{\boldsymbol{x}'} p(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{u}) \boldsymbol{\varphi}(\boldsymbol{x}') \right) + \eta\varepsilon + \lambda. \end{aligned} \tag{4.10}$$

We substitute $V_{\boldsymbol{v}}(\boldsymbol{x}) = \boldsymbol{v}^T \varphi(\boldsymbol{x})$. Differentiating the Lagrangian w.r.t $p(\boldsymbol{x}, \boldsymbol{u})$

$$\begin{aligned} \partial_{p(\boldsymbol{x},\boldsymbol{u})} L = r(\boldsymbol{x}, \boldsymbol{u}) - \eta \left(\log \frac{p(\boldsymbol{x}, \boldsymbol{u})}{q(\boldsymbol{x}, \boldsymbol{u})} + 1\right) - \lambda \\ - V_{\boldsymbol{v}}(\boldsymbol{x}) + \mathbb{E}_{p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})} \left[V_{\boldsymbol{v}}(\boldsymbol{x}')\right] = 0, \end{aligned} \tag{4.11}$$

and setting the derivative to zero yields the solution

$$p(\boldsymbol{x},\boldsymbol{u}) = q(\boldsymbol{x},\boldsymbol{u}) \exp\left(\frac{\delta_{\boldsymbol{v}}(\boldsymbol{x},\boldsymbol{u})}{\eta}\right) \exp\left(\frac{-\eta-\lambda}{\eta}\right) \tag{4.12}$$

with $\delta_{\boldsymbol{v}}(\boldsymbol{x},\boldsymbol{u}) = r(\boldsymbol{x},\boldsymbol{u}) + \mathbb{E}_{p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})}\left[V_{\boldsymbol{v}}(\boldsymbol{x}')\right] - V_{\boldsymbol{v}}(\boldsymbol{x})$. Given that we require $\sum_{\boldsymbol{x},\boldsymbol{u}} p(\boldsymbol{x},\boldsymbol{u}) = 1$, it is necessary that

$$\exp\left(\frac{-\eta-\lambda}{\eta}\right) = \left(\sum\nolimits_{\boldsymbol{x},\boldsymbol{u}} q(\boldsymbol{x},\boldsymbol{u}) \exp\left(\frac{\delta_{\boldsymbol{v}}(\boldsymbol{x},\boldsymbol{u})}{\eta}\right)\right)^{-1}. \tag{4.13}$$

Setting Equation (4.13) into Equation (4.12) yields the closed form solution for $p(\boldsymbol{x},\boldsymbol{u})$. Reinserting Equation (4.12) into the Lagrangian (4.10)[1]

$$g(\eta,\lambda) = \eta\epsilon + \eta + \eta\lambda = \eta\epsilon + \eta \log \exp\left(\frac{\eta+\lambda}{\eta}\right). \tag{4.14}$$

As we know from Equation (4.13) that λ depends on $\boldsymbol{v}$, we substitute (4.13) to get the formulation of the dual function, which depends on η and $\boldsymbol{v}$.

$$g(\eta,\boldsymbol{v}) = \eta\epsilon + \eta \log \sum_{\boldsymbol{x},\boldsymbol{u}} q(\boldsymbol{x},\boldsymbol{u}) \exp\left(\frac{\delta_{\boldsymbol{v}}(\boldsymbol{x},\boldsymbol{u})}{\eta}\right). \tag{4.15}$$

Optimizing the Dual-Function. The dual function is in *log-sum-exp* form and therefore convex in $\boldsymbol{v}$. As we have one inequality constraint in the original optimization problem, we also get an inequality constraint for the dual problem which requires that $\eta > 0$. Hence, for a given set of samples $(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]})$, we have to solve the following problem[2]

$$\begin{aligned} \min_{\eta,\boldsymbol{v}} \quad & \eta\epsilon + \eta \log \sum_{\boldsymbol{x}^{[i]},\boldsymbol{u}^{[i]}} \frac{1}{N} \exp\left(\frac{\delta_{\boldsymbol{v}}(\boldsymbol{x}^{[i]},\boldsymbol{u}^{[i]})}{\eta}\right), \\ \text{s.t:} \quad & \eta > 0. \end{aligned} \tag{4.16}$$

[1] It is easier to just insert Equation (4.12) into the $\log p(\boldsymbol{x},\boldsymbol{u})$ term of the Lagrangian. All other terms connected to $p(\boldsymbol{x},\boldsymbol{u})$ cancel out.

[2] For numerical accuracy, it is recommendable to subtract the maximum $\delta_{\boldsymbol{v}}$ inside the exp and add it again outside the log, i.e.,

$$g(\eta,\boldsymbol{v}) = \eta\epsilon + \max\delta_{\boldsymbol{v}} + \eta \log \sum_{\boldsymbol{x}^{[i]},\boldsymbol{u}^{[i]}} \frac{1}{N} \exp\left(\frac{\delta_{\boldsymbol{v}}(\boldsymbol{x}^{[i]},\boldsymbol{u}^{[i]}) - \max\delta_{\boldsymbol{v}}}{\eta}\right)$$

Any optimizer for constraint optimization problems can be used to solve this problem, for example fmincon in MATLAB. This optimization can typically performed more efficiently by providing the optimization algorithm also the derivatives of g, which are given by

$$\partial_\eta g(\eta, \boldsymbol{v}) = \epsilon + \log\left(\sum_i \frac{1}{N} Z_i\right) - \frac{\sum_i Z_i \delta_{\boldsymbol{v}}(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]})}{\eta \sum_i Z_i}, \tag{4.17}$$

$$\partial_{\boldsymbol{v}} g(\eta, \boldsymbol{v}) = \frac{\sum_i Z_i \left(\mathbb{E}_{p(\boldsymbol{x}'|\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]})}\left[\boldsymbol{\varphi}(\boldsymbol{x}')\right] - \boldsymbol{\varphi}(\boldsymbol{x}^{[i]})\right)}{\sum_i Z_i}, \tag{4.18}$$

with $Z_i = \exp\left(\delta_{\boldsymbol{v}}(\boldsymbol{x}^{[i]}, \boldsymbol{u}^{[i]})/\eta\right)$.

Dual-Function of Episode-based REPS. The derivation of the dual-function for parameter-based REPS follows the derivation given for the infinite horizon REPS. For this reason, we will only state the resulting dual-function for the contextual policy search setup and skip the derivation. The dual-function is given by

$$g(\eta, \boldsymbol{v}) = \eta\epsilon + \boldsymbol{v}^T \hat{\boldsymbol{\varphi}} + \eta \log \sum_{\boldsymbol{s}, \boldsymbol{\theta}} q(\boldsymbol{s}, \boldsymbol{\theta}) \exp\left(\frac{\delta_{\boldsymbol{v}}(\boldsymbol{s}, \boldsymbol{\theta})}{\eta}\right), \tag{4.19}$$

where $\delta_{\boldsymbol{v}}(\boldsymbol{s}, \boldsymbol{\theta}) = R(\boldsymbol{s}, \boldsymbol{\theta}) - \boldsymbol{v}^T \boldsymbol{\varphi}(\boldsymbol{s})$.

Dual-Function of Hierarchical REPS. The dual function of HiREPS is given by

$$g(\eta, \xi, \boldsymbol{v}) = \eta\epsilon + \hat{H}_q \kappa \xi + \eta \log \sum_{\boldsymbol{s}, \boldsymbol{\theta}} \tilde{p}(o|\boldsymbol{s}, \boldsymbol{\theta})^{1+\frac{\xi}{\eta}} \exp\left(\frac{\delta_{\boldsymbol{v}}(\boldsymbol{s}, \boldsymbol{\theta})}{\eta}\right), \tag{4.20}$$

where $\delta_{\boldsymbol{v}}(\boldsymbol{x}, \boldsymbol{u})$ is defined as for the episode-based REPS algorithm and ξ is the Lagrangian multiplier connected with the constraint which prevents an overlapping of the options.