Journal of Machine Learning Research 0 (0000) 0-00 Submitted 0/00; Published 00/00

Non-parametric Policy Search with Limited Information Loss

Herke van Hoof HERKE.VANHOOF@MAIL.MCGILL.CA
School of Computer Science, McGill University,
McConnell Eng Bldg, Room 318, 3480 University St, Montreal, Quebec, Canada

Gerhard Neumann GNEUMANN@LINCOLN.AC.UK
Lincoln Centre for Autonomous Systems, Lincoln University, Lincoln, United Kingdom

Jan Peters PETERS@IAS.TU-DARMSTADT.DE
Intelligent Autonomous Systems Group, TU Darmstadt, Darmstadt, Germany
Max Planck Institute for Intelligent Systems, Tübingen, Germany

Editor: ...

Abstract

Learning complex control policies from non-linear and redundant sensory input is an important challenge for reinforcement learning algorithms. Non-parametric methods that approximate values functions or transition models can address this problem, by adapting to the complexity of the dataset. Yet, many current non-parametric approaches rely on unstable greedy maximization of approximate value functions, which might lead to poor convergence or oscillations in the policy update. A more robust policy update can be obtained by limiting the information loss between successive state-action distributions. In this paper, we develop a policy search algorithm with policy updates that are both robust and non-parametric. Our method can learn non-parametric control policies for infinite horizon continuous Markov decision processes with non-linear and redundant sensory representations. We investigate how we can use approximations of the kernel function to reduce the time requirements of the demanding non-parametric computations. In our experiments, we show the strong performance of the proposed method, and how it can be approximated efficiently. Finally, we show that our algorithm can learn a real-robot underpowered swing-up task directly from image data.

Keywords: Reinforcement Learning, Kernel Methods, Policy Search, Robotics

1. Introduction

Learning continuous valued control policies directly from sensory input presents a major obstacle to applying reinforcement learning (RL) methods effectively in realistic settings. In such settings, there exist two major problems. First, the dimensionality of sensory inputs often makes discretization of the state-space infeasible. Secondly, in such settings, the amount of data that can be used to learn policies is often severely limited, increasing the bias in policy updates which can lead to oscillations or divergence (Mannor et al., 2007; Peters et al., 2008).

For the first problem, algorithms have been developed that rely on human-designed features for value function approximations or specialized parametric policies (Kaelbling et al., 1996; Kober et al., 2013; Bartlett, 2003). However, the applicability of such methods is limited in non-linear, redundant sensory domains where defining good features to linearly approximate value functions or policies is non-trivial. Alternative methods use deep neural networks to represent value functions or policies non-linearly (Mnih et al., 2015; Schulman et al., 2015; Lillicrap et al., 2016; Mnih et al., 2016). These representations have many parameters, and thus tend to require sampling many data points.

Recently, there has been a lot of progress towards avoiding the dependence on engineered features by using non-parametric techniques. Such techniques often use kernel functions to implicitly define a (possibly infinite) features space, replacing the manual definition of features. In contrast to task-specific hand-tuned feature spaces, many popular kernels are applicable to a large number of problems as the resulting representation can adapt to the complexity of the data. Non-parametric techniques have been successfully used in value-function methods, for example by Grünewälder et al. (2012b), Nishiyama et al. (2012), and Kroemer and Peters (2011). An overview of related work on non-parametric methods is given in Section 4.3 on page 31. Such methods generally require the inversion of matrices that grow with the number of data points, which limits their applicability. Another shortcoming is that these methods are still susceptible to the problem of data scarcity.

The problem of data scarcity is aggravated by the lack of a notion of the sampled data or sampling policy in many reinforcement learning approaches. At any point in the learning process, knowledge about the MDP tends to be represented in two ways. First, the outcomes of recently sampled state-action pairs are stored in memory. Secondly, the current sampling policy implicitly represents the state-action pairs sampled longer ago (as well as the initial policy). Any policy update that cause the policy to represent new samples better might cause the policy to represent older samples less well, thus losing information. Many methods update policies or value functions without regard to the sampling policy, which can result in a critical loss of essential information given that the recently sampled state-action pairs stored in, e.g., a mini-batch or replay memory are usually not sufficient to completely characterize the MDP.

As a result, choosing an improved policy purely based on sampled returns often yields fast but premature convergence to a suboptimal policy. Such updates favor biased solutions that eliminate states in which, by chance, only bad actions have been tried out. This problem is known as *optimization bias* (Mannor et al., 2007). Optimization biases may appear in both on- and off-policy reinforcement learning methods due to under-sampling (e.g., if we cannot sample sufficiently many of the state-actions pairs prescribed by a policy, we will over-fit), model errors or even the policy update step itself.

In an on-line setting, many methods address this problem implicitly by staying close to the previous policy. For example, policy gradient methods allow only small incremental policy changes. The Fisher information metric—that occurs in policy updates using the natural policy gradients (Kakade, 2002; Peters and Schaal, 2008)—can be seen as a Taylor expansion of the loss of information or *relative entropy* between the path distributions generated by the original and the updated policy (Bagnell and Schneider, 2003a). Instead of bounding this Taylor approximation, we can explicitly bound the relative entropy between successive state-action distributions, leading to our Relative Entropy Policy Search (REPS) algorithm[1]. We discuss related policy search methods that limit the information loss in Section 4.1 on page 28.

1. This paper draws from our earlier conference papers (Peters et al., 2010; van Hoof et al., 2015a,b).

In this paper, we propose a method based on this insight, that allows us to compute new policies given a data distribution both for off-policy or on-policy reinforcement learning. We start from the optimal control problem statement subject to the constraint that the loss in information is bounded. For continuous domains, where a suitable set of features is often not available, we develop a non-parametric version of this algorithm. This algorithm uses general kernels to define (possibly infinite) feature spaces implicitly, and consider ways to efficiently approximate this method to make it applicable to large datasets.

In our experiments, we show that our method outperforms other non-parametric methods on a reaching task and an underpowered swing-up task. We also show our method performs well compared to neural-network based methods on a variant of the puddle world task with a 400-dimensional redundant input representation. Furthermore, we evaluate different approximations to process larger data sets efficiently. Finally, we show that using such an approximation, a real-robot pendulum swing-up task can be learned from vision data.

1.1 Problem Statement and Notation

In a Markov decision process (MDP), an agent in state $\mathbf{s}$ selects an action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$ according to a (possibly stochastic) policy π and receives a reward $\mathcal{R}_{\mathbf{s}}^{\mathbf{a}} \in \mathbb{R}$. We will assume continuous state-action spaces: $\mathbf{s} \in \mathcal{S} = \mathbb{R}^{D_\mathrm{s}}$, $\mathbf{a} \in \mathcal{A} = \mathbb{R}^{D_\mathrm{a}}$. If the Markov decision process is ergodic, for each policy π, there exists a stationary distribution $\mu_\pi(\mathbf{s})$ such that $\int_{\mathcal{S}} \int_{\mathcal{A}} \mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \mu_\pi(s) \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} = \mu_\pi(\mathbf{s}')$, where $\mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}} = p(\mathbf{s}'|\mathbf{a}, \mathbf{s})$. The goal of a reinforcement learning agent is to choose a policy such that the joint state-action distribution $p_\pi(\mathbf{s}, \mathbf{a}) = \mu_\pi(\mathbf{s})\pi(\mathbf{a}|\mathbf{s})$ maximizes the average reward $J(\pi) = \int_{\mathcal{S}} \int_{\mathcal{A}} \pi(\mathbf{a}|\mathbf{s}) \mu_\pi(\mathbf{s}) \mathcal{R}_{\mathbf{s}}^{\mathbf{a}} \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}$.

The goal of relative entropy policy search is to obtain policies that maximize the expected reward $J(\pi)$ while bounding the information loss with respect to reference distribution $q(\mathbf{s}, \mathbf{a})$, i.e.,

$$\begin{aligned}
\max_{\pi, \mu_\pi} \quad & \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s}) \mu_\pi(\mathbf{s}) \mathcal{R}_{\mathbf{s}}^{\mathbf{a}} \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}, && (1)\\
\text{s. t.} \quad & \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s}) \mu_\pi(\mathbf{s}) \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} = 1, && (2)\\
\forall s' \quad & \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s}) \mu_\pi(\mathbf{s}) \mathcal{P}_{\mathbf{ss}'}^{\mathbf{a}} \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} = \mu_\pi(\mathbf{s}'), && (3)\\
& \mathrm{KL}(\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s}) || q(\mathbf{s}, \mathbf{a})) \leq \epsilon, && (4)
\end{aligned}$$

where Eqs. (1-3) specify the general reinforcement learning objective (Eq. 1) with the constraints that $\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})$ is a distribution (Eq. 2) and μ_π is the stationary distribution under policy π (Eq. 3). Equation (4) specifies the additional bound on the KL divergence, where

$$\mathrm{KL}(p(x)||q(x)) = \int p(x) \log(p(x)/q(x)) \mathrm{d}x.$$

The reference distribution q is usually set to the state-action distribution induced by previous policies, where the initial explorative policy is a wide, uninformed distribution such as a zero-centered Gaussian with a larger variance. In each iteration, the policy is adapted to maximize the expected reward while respecting the constraint on the KL divergence. Thus, as learning progresses, the policy typically slowly converges towards a deterministic reward-maximizing policy.

In this paper, we aim at developing a reinforcement learning algorithm applicable in continuous state-action MDPs with non-linear and redundant state representations. We assume the transition and reward models of the MDP are unknown. Furthermore, we will concentrate on infinite-horizon problems.

2. Stable Policy Updates for Stochastic Continuous MDPs

The relative entropy policy search optimization problem in Eqs. (1)-(4) defines controller updates that maximize the expected average reward under a bound on the information loss. However, for continuous systems with stochastic dynamics and non-parametric controllers, it is not straightforward to solve the optimization problem directly. In this section, we explain the steps to obtain a practical algorithm. First, we show how to find the dual of the optimization problem. Subsequently, we discuss how this problem can be solved for stochastic systems that are continuous and non-linear. After that, we discuss how to relax the assumption of ergodicity of the MDP by transforming the average reward MDP into a discounted reward MDP. Solving the optimization problem results in a new optimal policy that is, however, only defined on the current set of samples. Therefore, we discuss how the sample-based optimal policy can be generalized to the entire state space. Finally, we will discuss how to set the hyper-parameters of the different steps of our method with minimal manual tuning.

2.1 Finding the Dual Problem

To find the dual to the optimization problem in Eqs. (1)–(4), first, we formulate the Lagrangian. For every constraint, we introduce a Lagrangian multiplier. Because Eq. (3) represents a continuum of constraints, we introduce a corresponding continuous state-dependent Lagrangian multiplier $V(\mathbf{s})$. Instead of summing the contributions for each constraint in the Lagrangian, here, we have to integrate instead. Analogously to the sampling distribution $q(\mathbf{s}, \mathbf{a})$, we will write $p_\pi(\mathbf{s}, \mathbf{a}) = \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})$ for the proposed state-action distribution to keep the exposition brief. Therefore, the Lagrangian

$$
\begin{aligned}
L(p, \eta, V, \lambda) = &\iint_{\mathcal{A}\times\mathcal{S}} p_\pi(\mathbf{s}, \mathbf{a})\mathcal{R}_{\mathbf{s}}^{\mathbf{a}} \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} + \int_{\mathcal{S}} V(\mathbf{s}') \left(\iint_{\mathcal{A}\times\mathcal{S}} p_\pi(\mathbf{s}, \mathbf{a})\mathcal{P}_{\mathbf{s}\mathbf{s}'}^{\mathbf{a}} \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} - \mu_\pi(\mathbf{s}') \right) \mathrm{d}\mathbf{s}' \\
&+ \lambda \left(1 - \iint_{\mathcal{A}\times\mathcal{S}} p_\pi(\mathbf{s}, \mathbf{a}) \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} \right) + \eta \left(\epsilon - \iint_{\mathcal{A}\times\mathcal{S}} p_\pi(\mathbf{s}, \mathbf{a}) \log \frac{p_\pi(\mathbf{s}, \mathbf{a})}{q(\mathbf{s}, \mathbf{a})} \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} \right).
\end{aligned}
$$

Using the identity $\mu_\pi(\mathbf{s}) = \int_A p_\pi(\mathbf{s}, \mathbf{a})\mathrm{d}\mathbf{a}$, the Lagrangian can be re-shaped in the more convenient form

$$
L(p, \eta, V, \lambda) = \lambda - \mathbb{E}_{p_\pi(\mathbf{s},\mathbf{a})} [V(\mathbf{s})] + \eta\epsilon + \mathbb{E}_{p_\pi(\mathbf{s},\mathbf{a})} \left[\mathcal{R}_{\mathbf{s}}^{\mathbf{a}} - \lambda + \int_{\mathcal{S}} V(\mathbf{s}')\mathcal{P}_{\mathbf{s}\mathbf{s}'}^{\mathbf{a}} \mathrm{d}\mathbf{s}' - \eta \log \frac{p_\pi(\mathbf{s}, \mathbf{a})}{q(\mathbf{s}, \mathbf{a})} \right].
$$

To find the optimal p, we take the derivative of L w.r.t. p and set it to zero

$$\frac{\partial L}{\partial p_\pi(\mathbf{s},\mathbf{a})} = \mathcal{R}_\mathbf{s}^\mathbf{a} - \lambda + \int_\mathcal{S} V(\mathbf{s}')\mathcal{P}_{\mathbf{ss}'}^\mathbf{a} \mathrm{d}\mathbf{s}' - \eta \log \frac{p_\pi(\mathbf{s},\mathbf{a})}{q(\mathbf{s},\mathbf{a})} - \eta - V(\mathbf{s}) = 0.$$

Therefore, we obtain the new state-action distribution

$$p_\pi(\mathbf{s},\mathbf{a}) = q(\mathbf{s},\mathbf{a}) \exp\left(\frac{\mathcal{R}_\mathbf{s}^\mathbf{a} + \int_\mathcal{S} V(\mathbf{s}')\mathcal{P}_{\mathbf{ss}'}^\mathbf{a} \mathrm{d}\mathbf{s}' - V(\mathbf{s})}{\eta}\right) \exp\left(\frac{-\lambda - \eta}{\eta}\right). \tag{5}$$

The function $V(\mathbf{s})$ resembles a value function, such that $\delta(\mathbf{s},\mathbf{a},V) = \mathcal{R}_\mathbf{s}^\mathbf{a} + \int_\mathcal{S} V(\mathbf{s}')\mathcal{P}_{\mathbf{ss}'}^\mathbf{a} \mathrm{d}\mathbf{s}' - V(\mathbf{s})$ can be identified as a Bellman error. Since $p_\pi(\mathbf{s},\mathbf{a})$ is a probability distribution, we can identify $\exp(-\lambda/\eta - 1)$ to be a normalization factor

$$\exp(-\lambda/\eta - 1) = \frac{1}{\iint_{\mathcal{A}\times\mathcal{S}} q(\mathbf{s},\mathbf{a}) \exp(\delta(\mathbf{s},\mathbf{a},V)/\eta)\, \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}} = \frac{1}{\mathbb{E}_{q(\mathbf{s},\mathbf{a})} \exp(\delta(\mathbf{s},\mathbf{a},V)/\eta)},$$

which yields the policy

$$\pi(\mathbf{a}|\mathbf{s}) \propto q(\mathbf{s},\mathbf{a}) \exp\left(\frac{\delta(\mathbf{s},\mathbf{a},V)}{\eta}\right). \tag{6}$$

To obtain the dual function, we re-insert the state-action probabilities $p_\pi(\mathbf{s},\mathbf{a})$ in the Lagrangian[2]

$$\begin{aligned} g(\eta,V,\lambda) =& \lambda + \eta\epsilon + \mathbb{E}_{p_\pi(\mathbf{s},\mathbf{a})}\left[\delta(\mathbf{s},\mathbf{a},V) - \lambda - \eta \log \frac{p_\pi(\mathbf{s},\mathbf{a})}{q(\mathbf{s},\mathbf{a})}\right] \\ =& \eta\epsilon + \eta \log\left(\mathbb{E}_{q(\mathbf{s},\mathbf{a})} \exp(\delta(\mathbf{s},\mathbf{a},V)/\eta)\right), \end{aligned}$$

We typically do not know the sampling distribution q, as it depends on the unknown system dynamics. However, the expected value over q can be approximated straightforwardly by taking the average of samples $1,\ldots,n$ taken from q. Note that λ and q do not appear in the final expression of the dual function

$$g(\eta,V) = \eta\epsilon + \eta \log\left(\frac{1}{n}\sum_{i=1}^n \exp(\delta(\mathbf{s}_i,\mathbf{a}_i,V)/\eta)\right). \tag{7}$$

To compute the Bellman error δ, the transition distribution is required. As this distribution is generally not known, δ needs to be approximated. The dual function (7) depends implicitly on reference distribution q through the samples.

2.2 Solving the Dual Problem

To solve Eqs. (1)–(4), we need to find the Lagrangian parameters that minimize the dual function in Equation (7), i.e., $(\eta^*, V^*) = \arg\min g(\eta, V)$. V is a function with domain $\mathcal{S}$, hence, for continuous domains, we will surely over-fit without additional assumptions. One possibility would be to assume V a function linear in designed features, but good features are

2. A detailed derivation is given in Appendix A.

task-specific and often hard to define. We will therefore make the more general assumption that V^* is of the form

$$V^* = \sum_{\tilde{\mathbf{s}} \in \tilde{\mathcal{S}}} \alpha_{\tilde{\mathbf{s}}} k_{\mathrm{s}}(\tilde{\mathbf{s}}, \cdot), \tag{8}$$

for some set of states $\tilde{\mathcal{S}}$ and scalars α, and a chosen reproducing kernel k_{s}. In other words, we assume $V^* \in \mathcal{F}$ for a reproducing kernel Hilbert space (RKHS) $\mathcal{F}$ with kernel k_{s}. The kernel k_{s} implicitly defines a (possibly infinite dimensional) feature map $\boldsymbol{\phi}(\mathbf{s}) = k_{\mathrm{s}}(\mathbf{s}, \cdot)$ (Schölkopf et al., 1999). Such an implicit definition has the advantage that we do not need to explicitly compute a feature basis for V^* (Hofmann et al., 2008). Kernels are in most cases easier to choose than feature vectors as the complexity of V^* can grow with the amount of training data. In the final algorithm, we will only work with inner product of implicit features that can be computed using the kernel function, $\boldsymbol{\phi}(\mathbf{s})^T \boldsymbol{\phi}(\mathbf{s'}) = k_{\mathrm{s}}(\mathbf{s}, \mathbf{s'})$. Therefore, we do not need to explicitly represent the (possibly infinite dimensional) feature maps (Hofmann et al., 2008; Schölkopf et al., 1999).

Embedding the Transition Model. Since the transition model $\mathcal{P}^{\mathbf{a}}_{\mathbf{ss'}}$ in Eq. (5) is unknown, we need to approximate $\mathbb{E}_{\mathbf{s'}}[V(\mathbf{s'})|\mathbf{s}, \mathbf{a}]$. One way to do so would be to estimate the joint density $p(\mathbf{s}, \mathbf{a}, \mathbf{s'})$ using, e.g., kernel density estimation, and solving the integral $\int_{\mathcal{S}} V(\mathbf{s'}) p(\mathbf{s}, \mathbf{a}, \mathbf{s'}) d\mathbf{s'}$. However, kernel density estimation in $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ requires many training samples, and solving the integral generally requires a time-consuming numerical procedure. Instead, as suggested by Song et al. (2009), we can directly estimate a conditional operator $\mathbf{C}_{S'|S,A}$ that maps features $\boldsymbol{\psi}(\mathbf{s}, \mathbf{a})$ of state-action pairs to the expected value of implicit features $\boldsymbol{\phi}(\mathbf{s'}) = k_{\mathrm{s}}(\mathbf{s'}, \cdot)$. The resulting embedding $\boldsymbol{\mu}_{\mathbf{s'}|\mathbf{s},\mathbf{a}} = \mathbb{E}_{\mathbf{s'}}[\boldsymbol{\phi}(\mathbf{s'})|\mathbf{s}, \mathbf{a}]$ can subsequently be used to approximate functions in $\mathcal{F}$ as V. Using such embeddings avoids estimating the joint density and leads to good results even for high-dimensional data, and renders calculations of expected values over a function in $\mathcal{F}$ straightforward without numerical integration (Song et al., 2013).

In order to learn the conditional operator, we will use a kernel over the state-action space to implicitly define state-action features $\boldsymbol{\psi}(\mathbf{s}, \mathbf{a}) = k_{\mathrm{s}}(\mathbf{s}, \cdot) k_{\mathrm{a}}(\mathbf{a}, \cdot)$. Given a sample $\{(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}'_1), \ldots, (\mathbf{s}_n, \mathbf{a}_n, \mathbf{s}'_n)\}$, the empirical conditional embedding is defined as

$$\hat{\boldsymbol{\mu}}_{S'|s,a} = \hat{\mathbf{C}}_{S'|S,A} \boldsymbol{\psi}(\mathbf{s}, \mathbf{a}) = \boldsymbol{\Phi} \boldsymbol{\beta}(\mathbf{s}, \mathbf{a}), \tag{9}$$

$$\hat{\mathbf{C}}_{S'|S,A} = \boldsymbol{\Phi} (\mathbf{K}_{\mathrm{sa}} + l_C \mathbf{I})^{-1} \boldsymbol{\Psi}^T, \tag{10}$$

where $\hat{\mathbf{C}}_{S'|S,A}$ is a learned conditional operator that allows the computation of embedding strengths $\boldsymbol{\beta}(\mathbf{s}, \mathbf{a}) = (\mathbf{K}_{\mathrm{sa}} + l_C \mathbf{I})^{-1} \mathbf{k}_{\mathrm{sa}}(\mathbf{s}, \mathbf{a})$, as suggested by Grünewälder et al. (2012a,b). In this equation, l_C is a regularization coefficient, the matrices $\boldsymbol{\Psi} = [\boldsymbol{\psi}(\mathbf{s}_1, \mathbf{a}_1), \ldots, \boldsymbol{\psi}(\mathbf{s}_n, \mathbf{a}_n)]$ and $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\mathbf{s}'_1), \ldots, \boldsymbol{\phi}(\mathbf{s}'_n)]$ consist of implicit feature factors, whereas matrix $\mathbf{K}_{\mathrm{sa}} = \boldsymbol{\Psi}^T \boldsymbol{\Psi}$ and vector $\mathbf{k}_{\mathrm{sa}}(\mathbf{s}, \mathbf{a}) = \boldsymbol{\Psi}^T \boldsymbol{\psi}(\mathbf{s}, \mathbf{a})$ contain kernel function evaluations between pairs of data points.[3] The mean embedding $\hat{\boldsymbol{\mu}}_{S'|s,a}$ can be seen as a vector-valued kernel ridge regressor that maps $\mathbf{s}, \mathbf{a}$ pairs to the expected function $k_{\mathrm{s}}(\mathbf{s'}, \cdot)$ (Grünewälder et al., 2012a).

3. This means $[\mathbf{K}_{\mathrm{sa}}]_{ij} = k_{\mathrm{s}}(\mathbf{s}_i, \mathbf{s}_j) k_{\mathrm{a}}(\mathbf{a}_i, \mathbf{a}_j)$, $[\mathbf{k}_{\mathrm{sa}}(\mathbf{s}, \mathbf{a})]_i = k_{\mathrm{s}}(\mathbf{s}_i, \mathbf{s}) k_{\mathrm{a}}(\mathbf{a}_i, \mathbf{a})$.

Evaluation of V. In the next step, we want to use the conditional embedding to evaluate V. Conditional embeddings in reproducing kernel Hilbert spaces have the property that conditional expectations of functions in $\mathcal{F}$ can be calculated as inner product in the Hilbert space (Song et al., 2009). Since $V \in \mathcal{F}$, i.e., is of the form (8), the expected value of V can be approximated using the embedded distribution (Song et al., 2009; Grünewälder et al., 2012a,b), i.e.,

$$\mathbb{E}_{\mathbf{s}'}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}] = \left\langle V, \hat{\boldsymbol{\mu}}_{S'|s,a} \right\rangle_{\mathcal{F}} = \sum_{i=1}^{n} \beta_i(\mathbf{s},\mathbf{a}) V(\mathbf{s}'_i).$$

In the dual function g from Equation (7), V is now only evaluated at sampled states $\mathbf{s}_i$ and $\mathbf{s}'_i$. As we assumed $V \in \mathcal{F}$, the generalized representer theorem (Schölkopf et al., 2001) tells us that there is at least one optimum of the form (8) with $\tilde{\mathcal{S}}$ the set of all sampled states[4]. Consequently, $\mathbb{E}_{\mathbf{s}'}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}] - V(\mathbf{s}) = \boldsymbol{\alpha}^T \tilde{\mathbf{K}}_{\mathrm{s}} \boldsymbol{\beta}(\mathbf{s},\mathbf{a}) - \boldsymbol{\alpha}^T \mathbf{k}_{\mathrm{s}}(\mathbf{s})$, where $\tilde{\mathbf{K}}_{\mathrm{s}}$ is the Gram matrix with entries $[\tilde{\mathbf{K}}_{\mathrm{s}}]_{ji} = k_{\mathrm{s}}(\tilde{\mathbf{s}}_j, \mathbf{s}'_i)$, and $[\mathbf{k}_s(\mathbf{s})]_j = k_{\mathrm{s}}(\tilde{\mathbf{s}}_j, \mathbf{s})$.

Finding a Numerical Solution. The dual problem can now be restated in terms of η and $\boldsymbol{\alpha}$, as

$$\min_{\eta,\boldsymbol{\alpha}} g(\eta,\boldsymbol{\alpha}) = \eta\epsilon + \eta \log\left(\sum_{i=1}^{n} \frac{1}{n} \exp\left(\frac{\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})}{\eta}\right)\right), \quad \text{s.t.} \eta \geq 0, \tag{11}$$

$$\delta(\mathbf{s},\mathbf{a},\boldsymbol{\alpha}) = \mathcal{R}_{\mathbf{s}}^{\mathbf{a}} + \boldsymbol{\alpha}^T \left(\tilde{\mathbf{K}}_{\mathrm{s}} \boldsymbol{\beta}(\mathbf{s},\mathbf{a}) - \mathbf{k}_{\mathrm{s}}(\mathbf{s})\right). \tag{12}$$

This objective is convex, and since the analytic gradient and Hessian for this objective are straightforward to obtain[5], we employ second order optimization methods to find the optimal η and $\boldsymbol{\alpha}$. Jointly optimizing for η and $\boldsymbol{\alpha}$ is rather slow because of the need of a constrained optimization method. As proposed by Lioutikov et al. (2014), we use a coordinate-descent like approach where, iteratively, η and $\boldsymbol{\alpha}$ are minimized separately, keeping the value of the other variable fixed. Thus, a fast unconstrained convex optimization algorithm can be used to minimize $\boldsymbol{\alpha}$, whereas a more expensive constrained minimizer can be used for η. Minimizations are run for a fixed number of updates instead of to convergence, and every iteration is initialized at the result of the last optimization, to speed up the optimization. Iterations continue until the constraints are fulfilled within an acceptable tolerance.

If we choose kernels $\mathbf{k}_{\mathrm{s}}(\mathbf{s}_i,\mathbf{s}_j) = \tilde{\phi}(\mathbf{s}_i)^T \tilde{\phi}(\mathbf{s}_j)$ and $\mathbf{k}_{\mathrm{sa}}((\mathbf{s}_i,\mathbf{a}_i),(\mathbf{s}_j,\mathbf{a}_j)) = \mathbb{1}_{\{(\mathbf{s}_i,\mathbf{a}_i)\}}((\mathbf{s}_j,\mathbf{a}_j))$, we obtain the original REPS formulation (Peters et al., 2010) as a special case, with corresponding Bellman error

$$\delta(\mathbf{s},\mathbf{a},\boldsymbol{\alpha}) = \mathcal{R}_{\mathbf{s}}^{\mathbf{a}} + \boldsymbol{\alpha}^T(\tilde{\phi}(\mathbf{s}') - \tilde{\phi}(\mathbf{s})). \tag{13}$$

In these equations, $\mathbb{1}$ is the indicator function, $\tilde{\phi}$ is a set of hand-crafted features, and $\mathbf{s}'$ is the observed outcome of applying action $\mathbf{a}$ in state $\mathbf{s}$. Our generalization allows the selection of widely applicable kernels that do not depend on hand-crafted features. Furthermore, avoiding the identity kernel function over the state-action space allows efficient learning in stochastic systems.

4. A sketch of the proof following Schölkopf et al. (2001) is given in Appendix B.
5. The dual and its partial derivatives and Hessians are given in Appendix A, together with a proof of convexity of the dual function.

2.3 Ensuring a Stationary Distribution

The REPS formulation (Eqs. 1–4) assumes the existence of a stationary distribution. However, not all MDPs have a stationary distribution for every policy π. For systems that do have a stationary distribution, steady-state behavior might not be realizable for real systems that need to be started and stopped. Furthermore, transient behavior, such as the swing-up of a pendulum, might be of greater interest than steady-state behavior.

We can ensure the system has a stationary distribution that includes such transients by resetting the system with a probability $1-\gamma$ at each time step. The system is subsequently set to a state from the initial state distribution $p_1(\mathbf{s})$. In this case, the expected value of V at the next time step is given by

$$\begin{aligned}\mathbb{E}[V(\mathbf{s'})|\mathbf{s},\mathbf{a}] &= \int_{\mathcal{S}} \gamma \mathcal{P}^{\mathbf{a}}_{\mathbf{ss'}} V(\mathbf{s}') + (1-\gamma) p_1(\mathbf{s'}) V(\mathbf{s'}) \mathrm{d}\mathbf{s}', \\ &= \gamma \boldsymbol{\alpha}^T \tilde{\mathbf{K}}_{\mathrm{s}} \boldsymbol{\beta}(\mathbf{s},\mathbf{a}) + (1-\gamma)\boldsymbol{\alpha}^T \hat{\boldsymbol{\mu}}_{S1}, \end{aligned} \tag{14}$$

where $\hat{\boldsymbol{\mu}}_{S1}$ is the empirical (observed) embedding of the initial state distribution and $\mathcal{P}^{\mathbf{a}}_{\mathbf{ss'}}$ are the transition probabilities of the original MDP. Such a reset procedure enables learning by removing the impracticable requirement of infinite roll-out length. In this way, we obtain a discounted setting similar to that used in RL methods that optimize the accumulated discounted reward.

2.4 Generalization of the Sample-Based Policy

The parameters resulting from the optimization, η and $\boldsymbol{\alpha}$, can be inserted back in Equations (12) and (6) to yield the desired probabilities $\{p_\pi(\mathbf{s}_1,\mathbf{a}_1),\ldots,p_\pi(\mathbf{s}_n,\mathbf{a}_n)\}$ at the sampled $(\mathbf{s},\mathbf{a})$ pairs (where $p_\pi(\mathbf{s},\mathbf{a}) = \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})$ as before). Conditioning on the current state yields the policy to be followed in the next iteration. However, since states and actions are continuous, we need to generalize from these weighted samples to nearby data points. To this end, we want to find a generalizing stochastic policy $\tilde{\pi}(\mathbf{a}|\mathbf{s})$ conditioned on the observed policy samples.

We first consider parametric policies $\tilde{\pi}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta})$ linear in features $\boldsymbol{\phi}(s)$ with parameters $\boldsymbol{\theta}$. Later, we will generalize our results to non-linear kernels. We place a Gaussian prior over the unknown policy parameters, and choose a Gaussian noise model for the conditional over actions. Consequently, a Bayesian model is specified that will allow us to find a posterior over parameters $\boldsymbol{\theta}$

$$\begin{aligned} p(\boldsymbol{\theta}) &= \mathcal{N}(0, \alpha^{-1}\mathbf{I}), \\ \tilde{\pi}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta}) &= \mathcal{N}(\boldsymbol{\theta}^T\boldsymbol{\phi}(\mathbf{s}), \beta^{-1}\mathbf{I}). \end{aligned}$$

By conditioning on the sampled state-action pairs, we obtain the familiar update equation

$$p(\boldsymbol{\theta}|\mathbf{a}_1,\ldots,\mathbf{a}_n,\mathbf{s}_1,\ldots,\mathbf{s}_n) = z^{-1} p(\boldsymbol{\theta}) \prod_{i=1}^{n} \tilde{\pi}(\mathbf{a}_i|\mathbf{s}_i;\boldsymbol{\theta}) \quad (\mathbf{s}_i,\mathbf{a}_i) \sim p_\pi(\mathbf{a},\mathbf{s}).$$

Here, z^{-1} is a normalization factor. However, our samples are drawn from $q(\mathbf{a},\mathbf{s})$ and not $p_\pi(\mathbf{a},\mathbf{s})$. From a log transformation of the update equation, we see that we can use

Algorithm 1 Policy iteration with relative entropy policy search (REPS)

```
repeat
    generate roll-outs according to $\tilde{\pi}_{i-1}$
    minimize dual                                   $\eta^*, \boldsymbol{\alpha}^* \leftarrow \arg\min g(\eta, \boldsymbol{\alpha})$                      Eq. 11
    calculate Bellman errors for each sample        $\delta_j \leftarrow \mathcal{R}_j + \boldsymbol{\alpha}^{*T}\left(\tilde{\phi}(\mathbf{s}'_j) - \tilde{\phi}(\mathbf{s}_j)\right)$   Eq. 13
    calculate the sample weights                    $w_j \leftarrow \exp(\delta_j/\eta^*)$                                   Sec. 2.4
    fit a generalizing policy                       $\tilde{\pi}_i(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mu(\mathbf{s}), \sigma^2(\mathbf{s}))$                    Sec. 2.4
until convergence
```

importance sampling to estimate $\boldsymbol{\theta}$ using the approximation

$$\begin{aligned}\log p(\boldsymbol{\theta}|\mathbf{a}_1, \mathbf{s}_1, \ldots, \mathbf{a}_n, \mathbf{s}_n) &= \log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \log \tilde{\pi}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta}) + \text{const.} && (\mathbf{s}_i, \mathbf{a}_i) \sim p_\pi(\mathbf{a}, \mathbf{s}),\\ &\approx \log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \frac{p_\pi(\mathbf{a}, \mathbf{s})}{q(\mathbf{a}, \mathbf{s})} \log \tilde{\pi}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta}) + \text{const.} && (\mathbf{s}_i, \mathbf{a}_i) \sim q(\mathbf{a}, \mathbf{s}),\\ &= \log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \log \tilde{\pi}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta})^{\frac{p_\pi(\mathbf{a},\mathbf{s})}{q(\mathbf{a},\mathbf{s})}} + \text{const.} && (\mathbf{s}_i, \mathbf{a}_i) \sim q(\mathbf{a}, \mathbf{s}),\end{aligned}$$

As the new state-action distribution p_π is of the form given in (6), we can write the importance weights

$$w_i = \frac{p_\pi(\mathbf{s}_i, \mathbf{a}_i)}{q(\mathbf{s}_i, \mathbf{a}_i)} = \exp\left(\frac{\delta(\mathbf{s}_i, \mathbf{a}_i, V^*)}{\eta^*}\right).$$

Since $\tilde{\pi}(\mathbf{a}|\mathbf{s};\boldsymbol{\theta})$ is Gaussian in our model, raising to the power of w_i simply scales the variance by $1/w_i$. By exponentiating both sides again, and using the familiar procedure for weighted Bayesian linear regression (Gelman et al., 2004), we find the predictive distribution

$$\tilde{\pi}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\beta\boldsymbol{\phi}(\mathbf{s})^T\mathbf{S}_n\boldsymbol{\Phi}^T\mathbf{D}^{-1}\mathbf{A}, \boldsymbol{\phi}(\mathbf{s})^T\mathbf{S}_n\boldsymbol{\phi}(\mathbf{s}) + \beta^{-1}), \quad S_n = (\beta\boldsymbol{\Phi}^T\mathbf{D}^{-1}\boldsymbol{\Phi} + \alpha\mathbf{I})^{-1}, \tag{15}$$

where $\mathbf{D}$ is a diagonal weighting matrix with $\mathbf{D}_{ii} = 1/w_i$ and $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]^T$.

The policy mean is of the form of the lower bound introduced by Dayan and Hinton (1997), and the policy that maximizes this lower bound can be found through a weighted linear regression (Peters and Schaal, 2007), although that framework does not employ a Bayesian formulation and therefore cannot represent uncertainty in the parameters. Note that instead of basing the weights on a transformation of the reward function, our approach uses a transformation of the Bellman error, which takes the long-term expected rewards into account. The critical step of choosing the transformation was done by manual design in earlier work (Peters and Schaal, 2007; Dayan and Hinton, 1997)), while here the transformation directly results from the optimization problem (1)-(4). Algorithm 1 on page 9 shows how the different steps of our approach fit together in the special case of linear value functions and policy, and using sampled outcomes to approximate the Bellman error. This form of the algorithm was introduced in our earlier work (Peters et al., 2010).

Algorithm 2 Policy iteration with non-parametric REPS (NP-REPS)

```
repeat
    generate roll-outs according to $\tilde{\pi}_{i-1}$
    calculate kernel embedding strengths    $\boldsymbol{\beta}_j \leftarrow (\mathbf{K}_{\text{sa}} + l_C \mathbf{I})^{-1}\mathbf{k}_{\text{sa}}(\mathbf{s}_j, \mathbf{a}_j)$    Sec. 2.2
    minimize kernel-based dual    $\eta^*, \boldsymbol{\alpha}^* \leftarrow \arg\min g(\eta, \boldsymbol{\alpha})$    Eq. 11
    calculate kernel-based Bellman errors    $\delta_j \leftarrow \mathcal{R}_j + \boldsymbol{\alpha}^{*T}\left(\tilde{\mathbf{K}}_{\text{s}}\boldsymbol{\beta}_j - \mathbf{k}_{\text{s}}(\mathbf{s}_j)\right)$    Eq. 12
    calculate the sample weights    $w_j \leftarrow \exp(\delta_j/\eta^*)$    Sec. 2.4
    fit a generalizing non-parametric policy    $\tilde{\pi}_i(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mu(\mathbf{s}), \sigma^2(\mathbf{s}))$    Sec. 2.5
until convergence
```

2.5 Non-Parametric Generalizing Policies

For non-parametric policies, Eq. (15) can be kernelized straightforwardly, to yield

$$\tilde{\pi}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mu(\mathbf{s}), \sigma^2(\mathbf{s})), \quad \mu(s) = \mathbf{k}_{\text{s}}(\mathbf{s})^T(\mathbf{K}_{\text{s}} + l\mathbf{D})^{-1}\mathbf{A}, \tag{16}$$

$$\sigma^2(\mathbf{s}) = k(\mathbf{s}, \mathbf{s}) + l - \mathbf{k}_{\text{s}}(\mathbf{s})^T(\mathbf{K}_{\text{s}} + l\mathbf{D})^{-1}\mathbf{k}_{\text{s}}(\mathbf{s}), \tag{17}$$

where kernel vector $\mathbf{k}_{\text{s}}(\mathbf{s}) = \boldsymbol{\phi}(\mathbf{s})^T\boldsymbol{\Phi}$, kernel matrix $\mathbf{K}_{\text{s}} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$, and l is a free regularization hyper-parameter. Together with other hyper-parameters, such as the kernel bandwidth, l can be set by performing cross-validation on a maximum marginal likelihood objective.

Kober et al. (2011) derived a EM-based policy search approach that uses similar cost-sensitive Gaussian processes. The regularization term in these Gaussian processes is modulated with the inverse of the weight at each data point. However, there are some notable differences. First of all, in our case the weights w are found by transforming the advantage function rather than the reward function, allowing decision making in longer-horizon problems. Furthermore, that approach used a maximum likelihood perspective which allows derivation of the mean, but not the variance. In contrast, we derive our policy update using importance sampling from a Bayesian perspective that allows a principled derivation of the covariance update. Algorithm 2 shows how the different steps of our approach, non-parametric relative entropy policy search (NP-REPS), fit together.

2.6 Hyper-parameter Optimization

The computation of the conditional operator has open hyper-parameters, namely, the hyper-parameters of the kernels over $\mathbf{s}$ and $\mathbf{a}$ as well as the regularization parameter l_C. We set l_C and the hyper-parameters of kernel k_{a} through two-fold cross-validation on the objective

$$\sum_{i=1}^{n}\left\|\boldsymbol{\phi}(\tilde{\mathbf{s}}_i)^T\boldsymbol{\phi}(\mathbf{s}'_i) - \boldsymbol{\phi}(\tilde{\mathbf{s}}_i)^T\hat{\mathbf{C}}_{S'|S,A}\boldsymbol{\psi}(\mathbf{s}_i, \mathbf{a}_i)\right\|^2,$$

which minimizes the difference between actual embedding strength and the embedding strength predicted using the conditional operator $\hat{\mathbf{C}}_{S'|S,A}$ introduced in Eq. (10). This objective is based on the cross-validation objective proposed by Grünewälder et al. (2012a), but exploits the fact that the embedding will only be evaluated at known functions $\boldsymbol{\phi}(\tilde{\mathbf{s}})$.

The hyper-parameters of k_s, the kernel for the predicted variable $\mathbf{s}'$, cannot be tuned this way as trivial solutions exist. For example, essentially constant $\boldsymbol{\phi}(\mathbf{s}')$ with very high bandwidth minimize the prediction error. Instead, we set the hyper-parameters of k_s through minimization of the mean squared TD error in a two-fold cross-validation procedure. We choose this objective since minimizing the (residual) TD error is a common objective for feature selection (Parr et al., 2008) in reinforcement learning.

Separately from finding the hyper-parameters of the conditional operator, we optimize the hyper-parameters of the Gaussian process policy. The employed optimization objective is the weighted marginal likelihood, with weights w_i as discussed in Sec. 2.5. This objective is maximized in a cross-validation procedure where every roll-out is used as separate fold.

2.7 Efficient Approximations for Large Data Sets

The proposed algorithm requires inverting several $n \times n$ matrices, where n is the number of samples. If open hyper-parameters (such as regularization parameters or kernel bandwidths) need to be optimized, this inversion happens inside the optimization loop. As a consequence, learning becomes slow if more than approximately 5000 samples are used on current computing hardware. Especially for complex problems and problems requiring a high control frequency, such a soft limit can be prohibitive. In order to scale our method to larger problems, approximate methods with high time efficiency have to be considered. In this section, we will discuss two families of methods: sparsification of the kernel matrix, and approximation of the kernel function using stochastic features. Related work on efficient calculations for non-parametric reinforcement learning are discussed in Section 4.4 on page 32.

Sparsification Approaches. One way of scaling up kernel methods is to consider sparsifications, where a small number of pseudo-inputs are used rather than the full data set. Multiple sparsification schemes have been proposed, notably the likelihood approximation used in the projected latent variables (PLV) approach (Seeger et al., 2003) and the Bayesian derivation of sparse pseudo-input Gaussian processes (SPGPs) by Snelson and Ghahramani (2006) in the context of supervised learning.

Such sparsifications have been used in a number of RL algorithms (Engel et al., 2003; Xu et al., 2014; Jung and Polani, 2007; Xu et al., 2007; Lever and Stafford, 2015). Often, a quadratic program is optimized to learn the embedding strength of the data points in the active set m. This approach typically results in an approximation $k(\mathbf{x}, \mathbf{x}') \approx \mathbf{k}_m(\mathbf{x})^T \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}')$. In this equation, $\mathbf{k}_m(\mathbf{x})$ and $\mathbf{K}_{mm}$ are a vector and a matrix, respectively, of similarities to the active set of m data-points (Jung and Polani, 2007; Xu et al., 2007). The same effective kernel is used in the PLV approach (Seeger et al., 2003). Note that, effectively, a non-stationary kernel is obtained that is parametrized by the data points in the active set m (Jung and Polani, 2007; Xu et al., 2007).

The SPGP approach uses a very similar kernel, but includes a state-dependent regularization term. This term proves to be helpful in gradient-based hyper-parameter optimization (Snelson and Ghahramani, 2006). The covariance of output points is then given by $\mathbf{K}_{nm} \mathbf{K}_{mm}^{-1} \mathbf{K}_{mn} + l\mathbf{I} + \boldsymbol{\Lambda}$. In this equation, the n^{th} element on the diagonal of $\boldsymbol{\Lambda}$ is given by $k(\mathbf{x}_n, \mathbf{x}_n) - \mathbf{k}_m(\mathbf{x}_n)^T \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}_n)$. The matrix $\mathbf{K}_{nm}$ denotes a Gram matrix between all n

input points and the active subset of m data points, and k_m and $\mathbf{k}_m$ denote a scalar and vector of corresponding kernel values.

To derive a sparsification with the same type of effective kernel using the reward-dependent regularization terms introduced in Section 2.5, we can start with the regular update equation for the non-parametric policy mean in Equation (16),

$$\mu(\mathbf{x}) = \mathbf{k}_n(\mathbf{x})^T(\mathbf{K}_{nn} + l\mathbf{D})^{-1}\mathbf{y},$$

where $\mathbf{y}$ is the vector of all training outputs. We replace the occurrences of the kernel with the effective kernel

$$\mu(\mathbf{x}) = \mathbf{k}_m(\mathbf{x})^T\mathbf{K}_{mm}^{-1}\mathbf{K}_{mn}(\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{mn} + l\mathbf{D} + \mathbf{\Lambda})^{-1}\mathbf{y}.$$

Now, we can apply the Woodbury identity to obtain the update equation

$$\mu(\mathbf{x}) = \mathbf{k}_m(\mathbf{x})^T(\mathbf{K}_{mm} + \mathbf{K}_{mn}(l\mathbf{D} + \mathbf{\Lambda})^{-1}\mathbf{K}_{nm})^{-1}\mathbf{K}_{mn}(l\mathbf{D} + \mathbf{\Lambda})^{-1}\mathbf{y}.$$

The empirical conditional embedding (Eq. 10) can similarly be approximated using this effective kernel. Starting from the regular update equation for the non-parametric policy covariance in Equation (17), in similar fashion we can derive the predictive variance

$$\sigma^2(\mathbf{x}) = k(\mathbf{x},\mathbf{x}) - \mathbf{k}_m(\mathbf{x})^T(\mathbf{K}_{mm}^{-1} + (\mathbf{K}_{mm} + \mathbf{K}_{mn}(l\mathbf{D} + \mathbf{\Lambda})^{-1}\mathbf{K}_{nm})^{-1})\mathbf{k}_m(\mathbf{x}) + l.$$

Analogously, a cost-regularized version of the PLV approach can be obtained in a similar form, but omitting the Λ terms. As the proposed sparsification scheme depends on the inverse of $\mathbf{K}_{mm}$, numerical problems can potentially ensue if the active subset m is poorly chosen such that two almost-equal data points are present. To address this issue, we regularize $\mathbf{K}_{mm}$ where necessary.

Alternatively, we consider fitting a regular Gaussian process to a set of M inducing inputs with pseudo-targets given by weighted linear regression. For pseudo-outputs $\tilde{\mathbf{y}}$ of M sparse inputs, a Gaussian process would predict $\hat{\mathbf{y}} = \mathbf{K}_{nm}(\mathbf{K}_{mm} + l\mathbf{I})^{-1}\tilde{\mathbf{y}}$ at all (active and passive) inputs. Since we know the true outputs $\mathbf{y}$ for the training data, a maximum likelihood solution can be found using weighted linear regression, considering $\mathbf{K}_{nm}(\mathbf{K}_{mm} + l\mathbf{I})^{-1}$ as design matrix. This approach yields the update equations for the mean

$$\mu(\mathbf{x}) = \mathbf{k}_m\tilde{\mathbf{K}}_{mm}^{-1}\tilde{\mathbf{y}},$$

where $\tilde{\mathbf{K}}_{mm}^{-1}$ is a regularized inverse $(\mathbf{K}_{mm} + l\mathbf{I})^{-1}$. The inducing output values $\tilde{\mathbf{y}}$ are

$$\tilde{\mathbf{y}} = (\tilde{\mathbf{K}}_{mm}^{-1}\mathbf{K}_{mn}\mathbf{D}\mathbf{K}_{nm}\tilde{\mathbf{K}}_{mm}^{-1} + l_2\mathbf{I})^{-1}\tilde{\mathbf{K}}_{mm}^{-1}\mathbf{K}_{mn}\mathbf{D}\mathbf{y},$$

where l_2 is an additional regularization parameter. As the covariance does not depend on the inducing output at the training points, the standard update equation for Gaussian processes can be used in this case. The update equation for the mean corresponds to that of a Gaussian process with an effective kernel $k(x, x') = \mathbf{k}_m(x)^T\tilde{\mathbf{K}}_{mm}^{-1}\tilde{\mathbf{K}}_{mm}^{-1}\mathbf{k}_m(x')$.

Random Fourier Features. Instead of sparsification, that is, using the exact kernel function at a subset of data points, we might instead use an approximation of the kernel function at all data points. One such approach is proposed by Rahimi and Recht (2007), who define a distribution $p(z)$ over mapping functions z such that the inner products in sampled feature spaces are unbiased estimates of the kernel evaluation $k(\mathbf{x}, \mathbf{y}) = \mathbb{E}[z(\mathbf{x})^T z(\mathbf{y})] \approx \mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{y})$, where $\mathbf{z}_i(\cdot) \sim p(z)$. They propose two kinds of random features that obey this criterion, Fourier features and binning features. As the binning features are suitable only for kernels that solely rely on the L1 norm between data points, we will focus on the Fourier features in this section. These features have been used successfully in various classification and regression tasks (Rahimi and Recht, 2007; Hernández-Lobato et al., 2014; Lu et al., 2016).

For the Fourier features, we require the Fourier transform $\alpha p(\boldsymbol{\omega})$ of a stationary (shift-invariant) kernel $k(\mathbf{x}, \mathbf{y})$, where α is a normalization factor that ensures $p(\boldsymbol{\omega})$ is a probability distribution. The inverse Fourier transform is thus

$$k(\mathbf{x}, \mathbf{y}) = \alpha \int_{\mathcal{R}^d} p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^T(\mathbf{x}-\mathbf{y})} \mathrm{d}\boldsymbol{\omega} = 2\alpha \mathbb{E}[\cos(\boldsymbol{\omega}^T \mathbf{x} + b)\cos(\boldsymbol{\omega}^T \mathbf{y} + b)], \quad b \sim \mathcal{U}(0, 2\pi).$$

We can approximate this integral using samples $(\boldsymbol{\omega}_i, b_i)$ with $i = 1, \ldots, L$ and obtain

$$k(\mathbf{x}, \mathbf{y}) \approx 2\alpha \frac{1}{L} \sum_{i=1}^{L} \cos(\boldsymbol{\omega}_i^T \mathbf{x} + b_i) \cos(\boldsymbol{\omega}_i^T \mathbf{y} + b_i) = \mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{y}),$$

with $\mathbf{z}_i(\mathbf{x}) = \sqrt{2\alpha L^{-1}} \cos(\boldsymbol{\omega}_i^T \mathbf{x} + b_i)$. As the chosen number of samples L gets smaller, the approximation gets coarser but computations will get faster, as we will need to invert $L \times L$ covariance matrices.

In this article, we will use the squared-exponential (or Gaussian) kernel $k_{\mathrm{es}}(\mathbf{x}, \mathbf{y}) = \alpha \exp(-0.5||\mathbf{x} - \mathbf{y}||_{\Sigma^{-1}})$ with diagonal covariance Σ. This kernel has a corresponding Fourier transform $p_{\mathrm{es}}(\boldsymbol{\omega}) = \mathcal{N}(0, \Sigma)$. Furthermore, for periodic data we will use a periodic kernel $k_{\mathrm{p}}(\mathbf{x}, \mathbf{y}) = \alpha \exp(-2\sin^2(0.5|\mathbf{x} - \mathbf{x'}|)/\sigma^2)$. This kernel is equivalent to the Gaussian kernel on periodic features

$$k_{\mathrm{p}}(\mathbf{x}, \mathbf{y}) = k_{\mathrm{es}}([\cos(\mathbf{x}), \sin(\mathbf{x})]^T, [\cos(\mathbf{y}), \sin(\mathbf{y})]^T).$$

Therefore, suitable random features can be generated as

$$\mathbf{z}_i(x) = \sqrt{2\alpha L^{-1}} \cos([\cos(\mathbf{x}), \sin(\mathbf{x})]\boldsymbol{\omega}_i + b_i).$$

Products of Gaussian and periodic kernels can likewise be written as a single multivariate Gaussian kernel on appropriate features of the inputs and handled in a similar way. As a result, all matrix inversions are now performed on $L \times L$ rather than $n \times n$ matrices.

3. Experiments

We first evaluate our method first on a reaching task, and the underpowered pendulum swing-up task. Then, we will consider two tasks with redundant inputs: we compare different methods on a variant of the puddle-world task with a 400-dimensional input representation, and show the ability of the proposed method to learn a variant of the swing-up task on a real robot that has access only to camera images rather than joint angles. In this section, we will first discuss elements of our set-up that are the same across tasks. Subsequently, we discuss implementation specifics and results for each task separately.

3.1 Experimental Set-up

We assume a realistic exploration setup in which the agent cannot choose arbitrary state-action pairs. Instead, as shown in Alg. 2, from an initial state distribution our agent explores using its stochastic policy. After every 10 roll-outs, the model learner and the policy of the agents are updated. To bootstrap the model and the policy, the agent is given 30 roll-outs using a random exploratory policy initially. To avoid excessive computations, we include a simple forgetting mechanism that only keeps the latest 30 roll-outs at any time[6]. As each roll-out contains 49 time steps on average in the larger tasks (as episodes are reset with a constant probability after each step), most computations are performed on approximately 1500×1500 matrices.

After each update, in simulated runs, the learning progress is evaluated by running the learned policy on 100 roll-outs with a fixed random seed. This data is not used for learning. For every method, we performed 10 trials, each consisting of 20 iterations so that 220 roll-outs were performed per trial (30 initial roll-outs plus 10 per iteration). In real-robot runs we evaluate the learned policy on the training samples, and performed 6 trials of 10 iterations each. The model and policy are refined incrementally in every iteration.

3.2 Compared Methods

We compared learning progress of the proposed approach to that of various other approaches. On the one hand, we consider the non-parametric value-function based methods introduced by Grünewälder et al. (2012b) and Pazis and Parr (2011). On the other hand, we will compare to the performance of neural-network based methods introduced by Lillicrap et al. (2016) and Schulman et al. (2015) that are well-suited for redundant input data. We also compare to versions of REPS that use the sample-based model approximation introduced by Daniel et al. (2016), and one that uses a fixed feature set. The relationship of the proposed approach to earlier non-parametric reinforcement learning methods will be discussed in Section 4.3 on page 31.

Sample Based Model. REPS only needs $\mathbb{E}_{\mathbf{s}'}\left[V(\mathbf{s}')|\mathbf{s},\mathbf{a}\right]$ at observed state-action pairs $(\mathbf{s}_i, \mathbf{a}_i)$. Therefore, if the system is deterministic, this expectation is simply $V(\mathbf{s}'_i)$ at the observed values for $(\mathbf{s}_i, \mathbf{a}_i)$. In stochastic systems, this sample-based method is used as approximation.

Feature Based REPS with Fixed Basis Functions. Instead of the non-parametric form of V assumed in this paper, we can follow earlier work and define a fixed feature basis (Peters et al., 2010; Daniel et al., 2016). We choose to use a similar number of the same radial basis functions used in the non-parametric method, but distribute these according to a regular grid over the state-action space.

6. As a consequence, the reference distribution q is a mixture of the previous three state-action distributions in our experiments.

Approximate Value Iteration. In this approach by Grünewälder et al. (2012b), the value function is assumed to be an element of the chosen RKHS. The maximization of the Q function requires discretizing $\mathbf{a}$, for which we choose 25 uniform bins in the allowable range. A deterministic policy selects $\mathbf{a}^* = \arg\max Q(\mathbf{s}_i, \cdot)$, but this policy does not explore, yielding bad performance in on-policy learning. To obtain an exploration-exploitation trade-off we replace the maximum by the soft-max operator $a^* \propto \exp(Q(\mathbf{s}_i, \cdot)c/\text{stdev}(Q(\mathbf{s}_i, \cdot)))$. The free parameter c specifies the greediness of the exploration/exploitation trade-off on normalized Q values. In an **on-policy** scheme, the new policy is used to obtain samples for the next iteration.

As a comparison to this on-policy scheme, we also compare using a **grid** of state-action pairs as training data. For a dense grid, this method has a richer input than all other methods, as those other methods start with uninformed roll-outs from the initial-state distribution. In a real system, obtaining such a grid is often unrealistic as the state cannot be set arbitrarily.

Non-parametric Approximate Linear Programming. Pazis and Parr (2011) introduce a non-parametric method, NPALP, that assumes the value function is Lipschitz. This assumption allows the RL problem to be formalized as a linear program. This method assumes the dynamics are deterministic. A greedy policy is obtained that is optimal if all state-action pairs have been visited. Since visiting all state-action pairs is infeasible in continuous systems, we include exploration by adding Gaussian distributed noise to the action in a fraction ϵ of the selected actions.

Trust Region Policy Optimization. This method, introduced by Schulman et al. (2015), uses a bound on the expected Kullback-Leibler divergence between successive policies, inspired by a theoretical policy iteration algorithm that guarantees non-decreasing expected returns. The method, TRPO, requires a parametric policy to be defined, but has been shown to work well with deep neural-network policies that can be used in many different tasks thanks to their flexibility. We use the reference implementation provided in the RLlab framework (Duan et al., 2016). This TRPO implementation requires a fixed time horizon, so we evaluate this model on a modified version of the tasks where the episodes are of fixed length, equal to the expected episode length used for other methods.

Deep Deterministic Policy Gradients. Lillicrap et al. (2016) introduced a policy gradient algorithm, DDPG, that works with deterministic policies. This is an actor-critic method that uses neural networks for both the Q function and the policy. After each time step, the new experience sample is stored in an 'experience replay' buffer, a randomly sampled batch from this buffer is then used to calculate the policy gradient. This technique diminishes the problem that successive samples tend to be highly correlated. We again used the RLlab implementation (Duan et al., 2016) with fixed episode length.

Linear Least Squares Policy Iteration As an example of a linear method, we use LSPI (Lagoudakis and Parr, 2003). This method learns a Q function by minimizing temporal differences. We only include this method to show that linear methods in general do not suffice on our task with pixel inputs. As suggested by Lagoudakis and Parr (2003), we use this method off-line, in contrast to the other methods. Thus, all data used in this method has been gathered using the initial exploration policy.

3.3 Approximation Methods

We compare the approximation methods described in Sec. 2.7 to each other as well as to a naive baseline. This comparison is done across a range of different numbers of features and/or inducing inputs, to evaluate the trade-off between a better approximation quality and a faster computation time.

Sub-sampling as Baseline. For the sub-sampling baseline, we simply train the method as described in Section 2.2 with a subset of the data points, ignoring the points that are not in the random subset.

Sparse Pseudo-input Gaussian Processes (SPGP). As an approximation, we use the method proposed by Snelson and Ghahramani (2006) to approximate the Gaussian process policy. To incorporate the desirability of data points, we use the modifications proposed in Section 2.7. As there is no straightforward extension of this method to learn the Bellman error terms (Equation 12), in these steps of the algorithm we interpret the matrix $\mathbf{K}_{nm}$ introduced in Section 2.7 as a feature matrix and subsequently calculate the Bellman error terms using the feature-based formulation introduced by Peters et al. (2010). In contrast to that work, we do not use single sample roll-outs, but calculate the embedding strengths $\boldsymbol{\beta}$ (Sec. 2.2) using a linear kernel with the pseudo-features as input. Snelson and Ghahramani (2006) suggest choosing the active subset by maximizing the marginal likelihood. However, this maximization is a computationally intensive process, and for the value function approximation, there is no criterion available equivalent to the maximum likelihood criterion for the supervised learning set-up. Therefore, we choose random subsets when applying this method.

Projected Latent Variables (LPV). Additionally, we compare to the sparsification method proposed by Seeger et al. (2003). Like for the previous method, we interpret the $\mathbf{K}_{nm}$ matrix as features and use these to calculate embedding strengths and Bellman error terms.

Regression-based Sparse GPs. We also consider the regression-based sparse Gaussian Processes proposed in Section 2.7. Again, the $\mathbf{K}_{nm}$ matrix is interpreted as design matrix for calculation of the embedding strengths and Bellman error terms.

Fourier Transform Based Approximation The last method we consider is based on the work of Rahimi and Recht (2007), with the extension to desirability-weighted samples as described in Section 2.7. As this method approximates the policy using features, we can use these features to calculate embedding strengths and Bellman error terms in the feature-based formulation introduced by Peters et al. (2010).

3.4 Reaching Task Experiment

In the reaching task, we simulate a simple two-link planar robot. In this task, the agent's actions directly set the accelerations of the two joints. Each link is of unit length and mass, and the system is completely deterministic. The agent gets negative reinforcement $r(\mathbf{s}, \mathbf{a}) = -10^{-4}||\mathbf{a}||_2^2 - ||\mathbf{x} - \mathbf{x}_{\text{des}}||_2^2$ according to the square of the applied action and of the distance of its end-effector to the Cartesian position $\mathbf{x}_{\text{des}} = [0.5, 0]$. Note that actions are two dimensional and states are four dimensional (joint positions and velocities). The robot starts stretched-out with the end-effector at $\mathbf{x} = [0,\, 2]$. The maximum applied acceleration is 50ms^2. We use $\gamma = 0.96$, which resets roll-outs after 24 samples, on average.

We use the commonly used squared-exponential (or Gaussian) kernel for angular velocities $\dot{\boldsymbol{\theta}}$ and actions. This kernel is defined as $k_{\mathrm{se}}(\mathbf{x}_i, \mathbf{x}_j) = \exp(-(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{D}(\mathbf{x}_i - \mathbf{x}_j))$ (with $\mathbf{D}$ a diagonal matrix containing free bandwidth parameters). However, for the angles $\boldsymbol{\theta}$ we need a kernel that represents its periodicity. We chose the kernel $k_{\mathrm{p}}(x_i, x_j) = \exp(-\sum_d \sin((x_i^{(d)} - x_j^{(d)})/(2\pi))^2/\mathbf{l}_d^2)$, where $\mathbf{l}$ is a vector of free bandwidth parameters. Consequently, we obtain a complete kernel

$$k((\boldsymbol{\theta}_i, \dot{\boldsymbol{\theta}}_i, a_i), (\boldsymbol{\theta}_j, \dot{\boldsymbol{\theta}}_j, a_j)) = k_{\mathrm{p}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) k_{\mathrm{se}}(\dot{\boldsymbol{\theta}}_i, \dot{\boldsymbol{\theta}}_j) k_{\mathrm{se}}(\mathbf{a}_i, \mathbf{a}_j),$$

composed of three simpler kernel functions.

Feature-based REPS needs a grid over the complete state-action space. Due to difficulties with the six-dimensional state-action space, we omitted this method. The exploration parameter ϵ of the NPALP method was set to 0.1, with the standard deviation of Gaussian noise set to $30Nm^2$. The Lipschitz constant was set to 1 with the velocity dimensions scaled by 1/5 for calculating distances. The exploration parameter c of the approximate value iteration method was set to 1.5. These values were manually tuned to yield fast and consistent learning progress. For REPS, we use a KL bound ϵ of 0.5 in our experiments, as this value empirically yielded acceptably fast learning progress while keeping updates smooth enough on a range of learning problems.

Results of the Reaching Task. The results of the reaching task are shown in Figure 1. As this task is deterministic, the sample-based model is optimal and provides an upper bound to the performance we can expect to get. Since non-parametric REPS with model learning needs to iteratively learn the transition distribution, its convergence is slower. After inspection of individual trials, the wide variance seems to be caused by occasional failures to find good hyper-parameters for the state-action kernel.

The baseline methods NPALP and value iteration using RKHS embeddings obtain good performance after the couple of iterations. However, if we iterate performing roll-outs and policy updates after those first steps, these methods fail to improve the policy consistently. The grid based value iteration scheme fails in this case: due to memory limitations the maximum grid size we could use was $[5 \times 5 \times 5 \times 5 \times 2 \times 2]$ in the 6-dimensional space, which appears to be insufficient to learn the task.

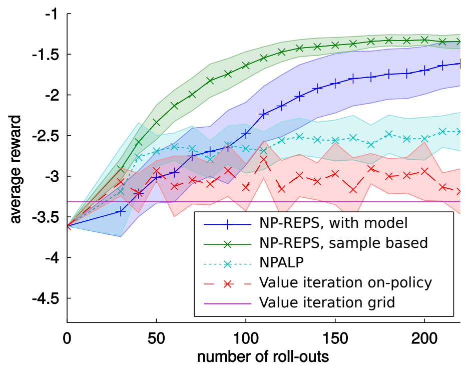
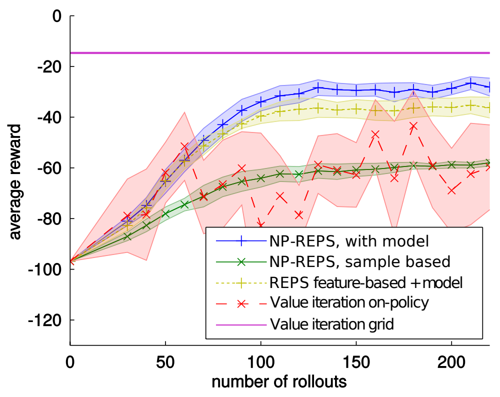
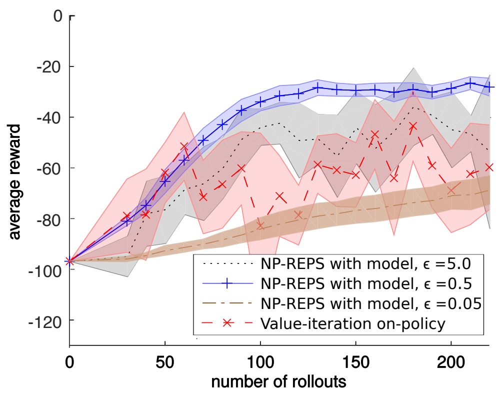


Figure 1: Comparison of learning progress of different methods on the reaching task. As this environment is deterministic, sample-based approximations of the transition functions are optimal. Error bars show twice the standard error of the mean. The value-iteration method does not depend on roll-outs, its performance is shown for comparison.



(a) Comparison of learning progress of different methods on the swing-up task.



(b) Comparison of learning progress under different bounds ϵ on the KL-divergence.

Figure 2: Results of the low-dimensional swing-up experiment. Our non-parametric relative entropy method outperforms the other on-policy learners. Error bars show twice the standard error of the mean.

3.5 Low-Dimensional Swing-up Experiment

In this experiment, we simulate a pendulum with a length of $l = 0.5$m and a mass $m = 10$kg distributed along its length. A torque a can be applied at the pivot. The pendulum is modeled by the dynamics equation $\ddot{\theta} = (glm \sin\theta + a - k\dot{\theta})/(ml^2/3)$, where $k = 0.25$Ns is a friction coefficient and $g = 9.81$ is the gravitational constant. The controller's sampling frequency is 20 Hz, i.e., every 0.05s the agent gets a reward and chooses a new action. The maximum admissible torque is 30Nm, which prevents a direct swing-up from the downwards position. Additive noise with a variance of $1/dt$ disturbs the controls, resulting in a standard deviation of about 4.5Nm per time step. The reward function was set to $r(s,a) = -10\theta^2 - 0.1\dot{\theta}^2 - 10^{-3}a^2$, where θ is mapped to $[-0.5\pi, 1.5\pi)$ to differentiate the rewards of clockwise and counter-clockwise swing-ups. We use a reset probability of 0.02 ($\gamma = 0.98$).

The algorithms directly access the state variable encoding the angle θ and the angular velocity $\dot{\theta}$, i.e., the state is defined as $\mathbf{s} = [\theta, \dot{\theta}]^T$. The same kernels are used as in the reaching task experiment (Sec. 3.4). The NPALP method was not designed for stochastic systems and is consequently omitted. We set the grid size for feature-based REPS to $[10 \times 10 \times 10]$ and for the value-iteration method to $[19 \times 11 \times 11]$. The greediness parameter c for on-policy value-iteration was set to 2 after manual tuning. We compare the KL bound ϵ used in the previous experiment, 0.5, to two extreme settings $\epsilon = 5$ and $\epsilon = 0.05$ to illustrate the effect of this bound.

Results of the Low-Dimensional Swing-up. We show a comparison of the discussed methods in Fig. 2a. The value iteration methods starts out competitively, but fails to keep improving the policy. Large variance indicates the learning process is unstable. The bounded policy update in REPS makes learning progress smooth by limiting information loss, limiting exploitation to retain more exploration in early stages of the learning process.

The sample-based baseline model performs considerably worse in this experiment, as it cannot account for stochastic transitions. The variant with fixed features performs well initially, but in later iterations, the non-parametric method is able to focus its representative power on frequently visited parts of the state-space, resulting in an improved performance.

We show a comparison for different value of the KL bound ϵ in Fig.2b. We plotted two rather extreme values for ϵ. For a large value of $\epsilon = 5$, we observe behavior that is similar to the greedy value iteration scheme. Initially, performance increases, but learning progress has a high variance and is not smooth. Suboptimal performance to both the proposed value $\epsilon = 0.5$ and the greedy value-iteration is caused by instability of the policy fitting described in Sec. 2.5 when the sample weights span several orders of magnitude. For a low value of $\epsilon = 0.05$, learning progress is very stable but slow. We found that setting $\epsilon = 0.5$ yielded good performance in all tasks we tried.

In contrast to the previous experiment, here, the grid-based value iteration method worked well, as shown in Fig. 2a. The policy learned by NP-REPS, shown in Fig. 3, sometimes overshoots the inverted position, which the grid based value iteration avoids. However, to reach this performance, a grid of training samples covering the full state-action space was needed. Providing such a grid is only feasible in simulation, as without an existing controller, it is generally not possible to start the dynamical system with arbitrary position and velocity. Furthermore, on higher-dimensional tasks, a grid of sufficient resolution would require impracticably many samples.

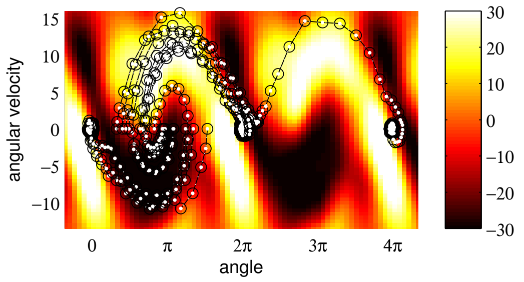


Figure 3: Mean of the learned stochastic policy. Superimposed are 15 trajectories starting at the x-axis between 0 and 2π. Most roll-outs reach the desired inverted pose, possibly after one swing back and forth. One roll-out overshoots and makes a full rotation before stabilizing the pendulum.

3.6 Image-based Puddle World Experiment

In this experiment, we investigated a modified version of the classical puddle-world task with a 400-dimensional redundant representation. In the puddle world task, the agent has to navigate to one corner of a two-dimensional world while avoiding high-cost areas ('puddles'). The aim of this experiment was to confirm the ability of our proposed method to handle such representation, and to compare our method to other method that have been shown to work well with raw sensory representations. We mostly follow the details for the puddle world described in Sutton (1996). As size of the goal region we use the area above the line $x + y = 1.8$. We specify the agent location by a simple rendered pixel image, where the agent is represented by a circle of 8% of the image size (the agent does not 'see' the puddles, but has to learn where they are from experience). We found that for the policy search methods we studied, a relatively low discount factor ($\gamma = 0.96$) or corresponding reset probability (0.04) provided lower-variance results. Such a discount factor makes long-term rewards less salient, thus we modified the per time step reward to -10. To make the task sensible in an average-reward setting, we do not restart when the goal area is reached, rather requiring the agent to learn to maintain this desired position.

A Gaussian kernel directly on pixel images would be very sensitive to even small shifts. A better choice is a kernel exponential in the squared difference between low-pass filtered images. To this end, we convolve the image with a Gaussian kernel of 20% of the image size. The same representation is also used for the other methods. Note, that methods based on neural networks would be able to learn this and other invariances, but this additional expressive power comes at the cost of many parameters that have to be learned. The image is finally downscaled to 20×20 pixels to be able reduce the size of a batch of images in working memory. This does not change much for non-parametric methods (as relative distance between images should stay roughly constant), and reduces the number of weights in neural network based methods. We again performed 10 separate trials for each method, using 15 iterations for REPS and an equivalent amount of training experiments for the other methods. In each iteration, 20 roll-outs were performed, retaining the roll-outs from the last 3 iterations in memory as before.

The methods under comparison were tuned as follows. For NP-REPS, we set the bandwidth of the value function to half the maximal pixel intensity and added a $l2$ regularizer $10^{-9}\boldsymbol{\alpha}^T\boldsymbol{\alpha}$ to Eq. (11) to avoid overfitting as this yielded good empirical performance. Other hyper-parameters were optimized as explained previously, with the bandwidth of all pixels for the model and policy tied together to speed up optimization. The KL-bound ϵ was again set to 0.5. For LSPI, we discretized the actions in horizontal and vertical directions to three values: the maximum in either direction or 0.

For the neural-network based methods, we started from the default values in the RLlab implementation (Duan et al., 2016). We adapted the batch size / epoch length to the 20 roll-outs performed per iteration. We manually tuned other parameters to maximize empirical performance. This procedure resulted in a KL bound of 0.05 for TRPO (much higher than the default, which is tuned for situations where a lot of data is available but the task is harder). For DDPG, we obtained a learning rate of 10^{-4} for both the Q function and policy, a minimum replay memory size of 1500 and a maximum memory size of 5000 and a batch size of 64 (the relatively high batch size made the method more sample efficient). For DDPG, we used a Gaussian exploration strategy with a decay period of 5000, as well as multi-layer perceptron for the Q function. We tried several convolutional and three-layer feedforward architectures, and got best performance for two layers with 4 hidden units each for TRPO and two layers with 8 hidden units for DDPG.

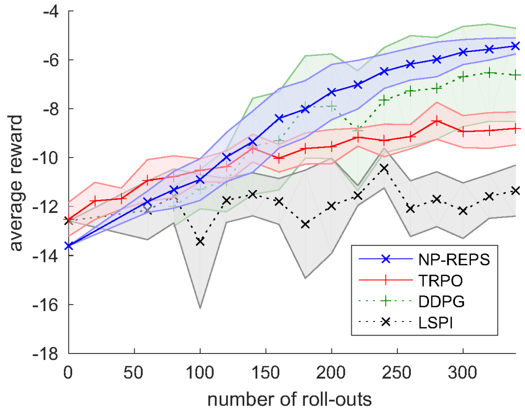


Figure 4: Results of the image-based puddle-world experiment. The RLlab version of DDPG only reports performance once the replay memory is initialized, so this graph starts after 80 roll-outs. Error bars show twice the standard error of the mean.

Results of the Image-based Puddle World Experiment. The performance of the compared methods is shown in Fig. 4. The poor performance of the LSPI method directly on raw pixels indicates that the problem is non-linear. TRPO with a neural network policy learns relatively slowly (we tried higher learning rates, however, these made learning more instable but did not yield higher average returns). In our experiments, TRPO needed at least 1000 roll-outs to converge to a good solution. DDPG with a neural network policy and Q function learns about as fast as NP-REPS, but yields high-variance results that tend to be slightly worse by the end of the experiment.

We want to stress two potential limitations of this experiment. First, difference between the compared methods might be due to either the representation of the policy and/or value function or the algorithm itself. Where NP-REPS uses kernel methods to represent the V function and the policy, TRPO and DDPG use neural networks. LSPI used a value function linear in the raw pixels with a greedy policy. Where in this experiment a non-parametric representation seems beneficial, in domains with visual distractors representations using neural networks might do better. Secondly, this experiment evaluates how well TRPO and DDPG do in a situation with a low number of training rollouts, whereas these methods are commonly used to work with a much higher number of roll-outs where they tend to do well.

3.7 Approximation Experiment on the Swing-up Task

We evaluated different schemes for approximating the kernel matrix on the pendulum swing-up task described in Section 3.5. We do this by training policies using the various methods discussed in Section 3.3. The trade-off between accuracy and computational speed is defined by the number of inducing inputs for the sparse kernel methods, or by the number of random features used in the Fourier-transform based approximation. The number of features (respectively, inducing inputs) was varied to investigate the trade-off between approximation accuracy and computation speed. We measured the learning progress using the average reward, and additionally logged the time needed per iteration for each condition. We report the average time needed over 10 trials and 10 iterations per trial.

Results and Discussion of the Approximation Experiment. A comparison of our results using different numbers of features is shown in Figure 5. In Figure 5a, all available inputs were used, and hence no approximation is needed. Thus, the sub-sampling method reduces to the original methods, and performs best. However, if the number of available bases is reduced to 500 (about one third of the available training points), just training on a sampled subset performs very badly, as shown in Figure 5b. In this set-up, the approximation based on a Fourier transform of the kernel seems the most suitable, both in terms of sample efficiency and of asymptotic value.

In Figure 5c, the available number of bases is reduced even more, to about 7% of the available training data. In this case, the regression-based sparsification method allows faster learning than the Fourier-based methods and both other sparsification methods, possibly because we used induced inputs $\tilde{y}$ that were optimal in a least-square sense. Considering that only 7% of the basis is available, the drop in performance relative to the full availability is modest.

Overall, the SPGP and LPV sparsification methods perform very similarly and yield similar asymptotic values as the regression-based sparsification methods. The main difference between these methods is the additional regularization term based on the approximation accuracy that Snelson and Ghahramani (2006) introduced. One of the main benefits of this additional regularization is a better behavior of the marginal likelihood for the purpose of optimizing the inducing input points (Snelson and Ghahramani, 2006). As we did not optimize for these points (as that would be too computationally costly to do inside the reinforcement-learning loop), our results seems plausible in this respect.

An indication of the time requirement of the different methods is given in Figure 5d. The implementations of the algorithms are not directly comparable, so drawing hard conclusions about the algorithms is not possible on the basis of this data. One implementation issue is that cross-validation for the subsampled and Fourier-based methods can be implemented using a fast decremental update of the matrix inverse. For the other sparsification methods, there is no straightforward way to implement this speed-up so these were trained using a 2-fold cross-validation setup (rather than using one fold for each trajectory, which would have been even more expensive computationally). Therefore, these three methods ('SPGP', 'LPV', and 'Sparse') are slowest in cases with many available inducing inputs.

When all data is available, applying any approximation takes more time than the baseline 'sampling' method. However, the 'Fourier' method tends to be much faster when the number of features is reduced—one of the reasons for this effect is that in our implementation, calculated features are cached. This caching benefits feature-based approximations, but not kernel-based ones. Compared to using all available bases without approximating, the 'Fourier' method with 500 bases is about 2 times as fast at a similar level of performance.

3.8 Real-Robot Swing-up Experiment from Vision Data

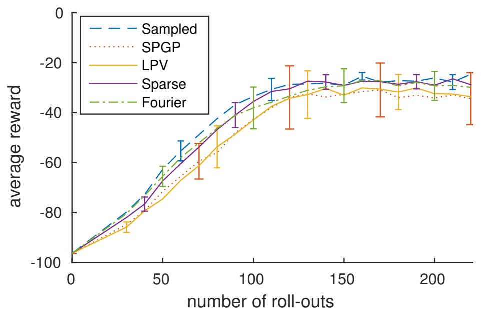
In this last experiment, we aim to validate our method on a real-robot task. We consider two scenarios: the learner either has access to the joint angles and velocity, or only to a redundant image-based representations of the state obtained through a camera pointed at the robot. Considering the time it takes to perform robot evaluations, and the wear to the robotic system, we provide a proof-of-concept of the proposed method without comparing to other methods.

Robot Set-up and System Dynamics. To the end-effector of a Kuka light-weight robot arm, we attached a wooden rod roughly 1kg heavy and 80 cm long. Red cardboard was used to make the pendulum visually salient. We aim to swing this rod up using the last degree-of-freedom, and limit the torque of the corresponding motor such that the pendulum cannot be swung up directly from the downwards position. This set-up is shown in Figure 6a.

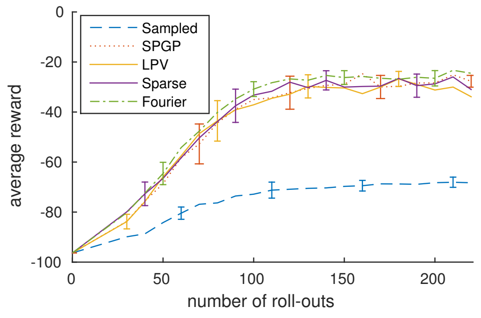
A couple of characteristics of the robot system make the task different from the simulated pendulum swing-up task in the previous section. The robot link is limited in its range of motion to less than 2π and in its velocity to less than about $4.2/s$, whereas the simulation had no such limitation. To prevent the robot from moving hard into these limits, we add a feedback signal that stops the robot if it gets close to those limits for safety[7]. The joint limits are illustrated in Figure 6b.

To perform exploration on the real robot, another issue to address is that high jerk might damage the gearboxes. Therefore, instead of controlling the torque as in the previous

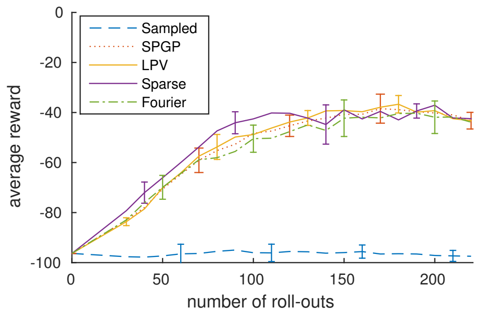
7. The exact feedback signals used are given in Appendix C.



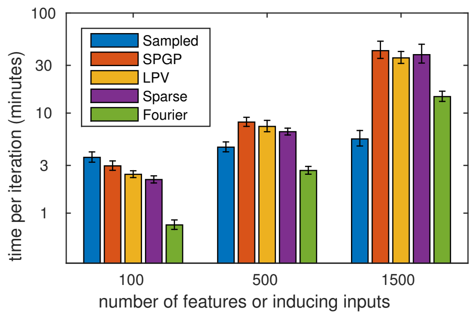
(a) Different approximation methods using 1500 features or all available (on average 1500) inducing inputs.



(b) Different approximation methods using 500 features or inducing inputs.



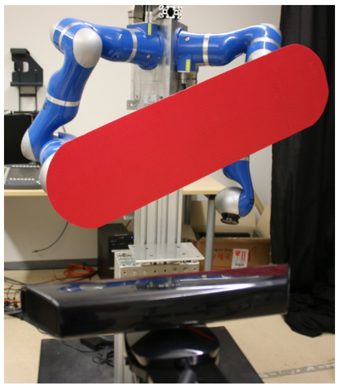
(c) Different approximation methods using 100 features or inducing inputs.

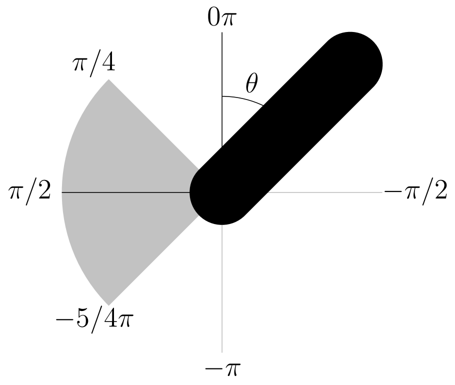


(d) Time requirement of different approximation methods on a 2.7 GHz processor running in single threads, log scale.

Figure 5: Results of the evaluation of different approximation methods using different numbers of features. In all graphs, error bars indicate twice the standard error of the mean. The learning curves show error bars only for every fifth iteration to keep the figures interpretable.

At a medium number of features (b), the Fourier-based random features deliver performance that is almost equal to the original algorithm, while requiring fewer computational resources (d). More computational resources can be saved by using only 100 features or inducing inputs, at the cost of a larger performance gap to the original algorithm (c).


(a)



(b)

Figure 6: The set-up of the real-robot swing-up experiment with redundant representation. (a) The robot set-up. One of the robot arms holds a pendulum which has a mass of about 1 kg and a length of about 80 cm. The Kinect camera in the foreground is used to provide feedback to the robot (proprioceptive joint information is not available to the robot). (b) An illustration of the set-up that illustrates the coordinate system used. As the image is shown from the camera's point of view, the coordinate system is inverted (clockwise θ). In the shaded area, an additional torque is applied to keep the robot from running into the joint limit at $\pi/2$.

experiments, we will control an increment to the torque. This increment will smoothly be added to the previous torque over the course of one time step. Controlling the increment, rather than directly controlling the torque, has the additional benefit that the applied torque tends to be more consistent over time, preventing 'washing out' of high-frequent control signals on the robot system. To preserve Markov properties, the previous torque has to be appended to the state vector. To provide comparable results over the different robot trials, instead of resetting the system randomly, we reset the system every 50 time steps to a set of ten different starting angles. The starting velocity and acceleration are set to zero.

Camera Set-up and Image Processing. The camera provides video frames at a rate of about 30Hz, but this rate can vary slightly during the experiments. To prevent synchronization issues between the camera and the control, we let the arrival of camera images govern the time step length. Since we want to control the robot at about ten Hertz, we choose a new action whenever every third camera image is received. Consequently, not all torques are applied for the same duration, providing a source of transition noise.

To keep the amount of storage space and processing time limited, we down-sample the images to 15×20 pixels and reduce the image to a single channel (by subtracting the average brightness from the red channel value, since the color of the pendulum is red). The image is blurred slightly with a low-pass filter to smooth out sensor noise, using a Gaussian kernel with a bandwidth of 6% of the image width. To provide the learner with a notion of velocity, the 15×20 pixel image and the 15×20 difference image to the previous time

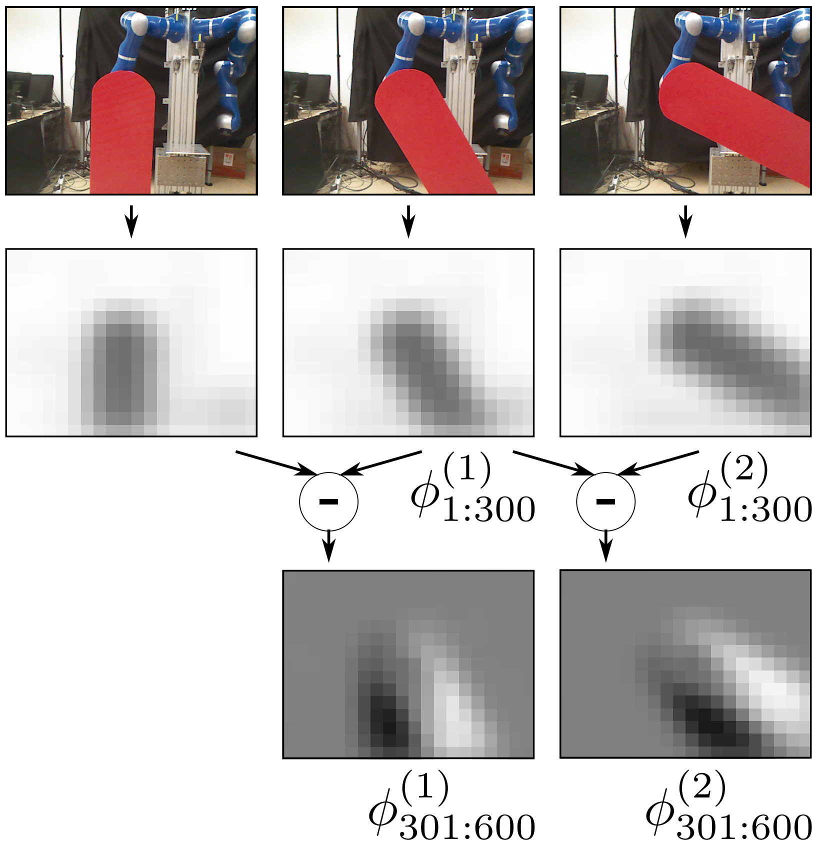


Figure 7: Sensor representation used by the robot. The Kinect camera image is down-sampled into 15 by 20 pixels and converted to a single-channel image by subtracting the average intensity from the red channel values. From the resulting sequence of images, the current image and the difference between the current image and the previous image are concatenated with the previously applied torque, yielding a 601-dimensional feature vector.

step were given to the robot as a concatenated 600-dimensional feature vector, as shown in Figure 7.

Employed Reward Function and Kernel. In contrast to the earlier experiment, the real robot system is not periodic as it cannot turn the joint more than 2π. Consequently, we need a reward function that punishes the robot for getting too close to the joint limit. We use the reward function

$$r(\mathbf{s}, a) = \exp\left(-(1.5\theta)^2\right) - 2(\theta - 0.25)\,\mathbb{1}_{\{x\in\mathbb{R}:x>0.25\}}(\theta) - 0.0005a^2,$$

where the first term rewards the robot for being close to the upright position and the second term punishes the robot for being too close to the joint limit. All else being equal, we prefer solutions that do not needlessly change the torques suddenly, as such behavior causes high jerk that can damage the robots. Therefore, the third term punishes large control actions.

As kernel on the state variables, we used a Gaussian kernel with three separate bandwidth parameters: one for all pixels of the current image, one for all pixels of the difference image, and one for the previously applied torque. The bandwidth parameters for all pixels of each image were tied together in this way to keep the hyper-parameter optimization manageable. For the learned transition model, the kernel on the action (applied jerk) is again a squared exponential.

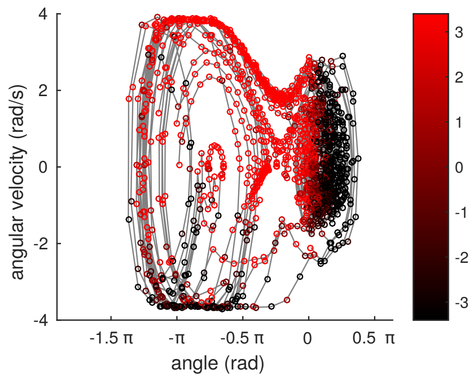
In this experiment, the heuristic for setting the bandwidth for the value function approximation according to the TD-error did not always work. Therefore, these three bandwidth parameters were set by hand, to 0.5, 1.4, and 4.0 respectively, as these values allowed good prediction performance while being able to represent the value function with enough accuracy. The bandwidth parameters and other kernel parameters for the learned transition model and the policy were set automatically by maximizing the same cross-validation objectives used in the previous experiments. The bound on the KL divergence was set to $\epsilon = 0.8$.

We used the Fourier feature approximation to the full kernel matrix, as the simulation experiments showed these features to yield good performance with an intermediate number of bases while also requiring relatively little computation time. Furthermore, computing the features and evaluating the linear Bayesian policy is straightforward to implement on a robotic system and can run in real-time as it has relatively low computational demands. We used 1000 random basis features, as the system is intrinsically higher dimensional compared to the simulated pendulum swing-up experiment (as we appended the previously applied torque to the state vector).

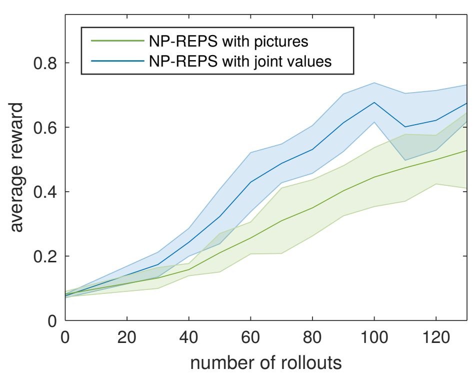
Sometimes, a feasible solution to the dual optimization problem could not be found. We considered this effect could be due to over-fitting to the sensor noise in the camera that made the system partially observable. We addressed this issue by projecting the 1000-dimensional feature vectors on all principal components with principal values at least 5% of the maximum principal value for purpose of optimization of the dual function (Eq. 11) only. This procedure yielded between 100 and 200 components on our dataset and addressed the problem satisfactorily. The dimension reduction also sped up the optimization of the dual function. Models and policies were still learned in the original 1000-dimensional spaces, as these steps contain regularization terms that make them robust to such over-fitting.

Results for the Real-robot Swing-up from Vision Data. The results of swing-up tasks with visual state features are shown in Figure 8. Of the six trials we performed, five resulted in policies that successfully swing-up and balance the pendulum. An example of the phase-space is shown in Figure 8a. It is apparent that the pendulum oscillates near the target position which occasionally causes the pendulum to drop. The pendulum-camera system has some system delay which could cause such oscillations, and which also prevents a finer time discretization.

Figure 8b shows the average learning progress for the task. The system learns from an initial uninformed policy that hardly obtains high rewards, with the fastest learning progress obtained between 40 and 100 roll-outs. Our result shows that non-parametric methods are a promising approach to dealing with redundant state representations such as images, since the task is successfully learned. Nevertheless, knowing a good representation, such as joint values, still resulted in better learning performance. Generally, many different representa-



(a) Phase-space trajectories with torque.



(b) Learning progress over six trials.

Figure 8: Results of the real-robot experiment from vision data. (a) Phase-space trajectories of one of the learned policies. Color indicates the applied torque (Nm). In most trajectories, the robot manages to swing up the pendulum and balance it around the upright position ($\theta = 0$). Near this position, the pendulum tends to oscillate as a consequence of time discretization and system delays. (b) Learning progress with input from either joint angles or pictures only. The REPS algorithm would converge to a locally optimum solution regardless of input modality, but seems to need more training time to reach such a local optimum using visual input. The graph shows the average reward over six independent trials. Error bars show twice the standard error.

tions could be used as long as a kernel can be defined that yields appropriate similarity values: in the end, the algorithm only performs operations based on those similarity values. Some pixels will never change their value. Such pixels are not problematic, as stationary kernels such as the squared exponential are not influenced by them.

4. Related Work

Over the years, many different reinforcement learning (RL) algorithms have been proposed, as reviewed by, among others, Bertsekas and Tsitsiklis (1996); Szepesvári (2010); Sutton and Barto (1998); Busoniu et al. (2010); Powell (2007); Bartlett (2003); Kaelbling et al. (1996); Wiering and Otterlo (2012) and Deisenroth et al. (2013). Many of the most well-known algorithms are value-function methods. However, there is no notion of the sampled data or sampling policy in the value function, making it impossible to limit the loss of information as the policy is updated (Peters et al., 2010). Furthermore, in continuous state-action spaces, such methods need to be approximated, which means that policy iteration does not necessarily improve the policy (Kakade and Langford, 2002; Bartlett, 2003).

Policy search methods are a complementary class of reinforcement learning approaches, which explicitly represent the sampling policy (Deisenroth et al., 2013). Such methods might additionally represent a value function, e.g. (Williams, 1992; Peters and Schaal, 2008; Sutton and Barto, 1998), in which case they are referred to as actor-critic algorithms. Policy search methods allow the learning agent to take the sampled data or the sampling policy into account. Other advantages include that policies might be easier to represent than value functions and convergent algorithms for policy search are known (Bagnell and Schneider, 2003a). Furthermore, for certain policy parameterizations, stability guarantees can be given (Deisenroth et al., 2013). If prior knowledge or task demonstrations are available, these can usually be integrated straightforwardly in policy search methods (Deisenroth et al., 2013; Peters and Schaal, 2006). For these reasons, policy search method have in practice proven to work well on real (e.g. robotic) systems (Deisenroth et al., 2013).

In high-dimensional domains, both value-function and policy search methods have often relied on hand-crafted feature representations (Kaelbling et al., 1996; Kober et al., 2013; Bartlett, 2003). This hand-tuning can largely be avoided by learning a suitable representation (Jonschkowski and Brock, 2015; Böhmer et al., 2013). Such an approach, however, often requires a lot of training data and usually relies on non-convex optimization. Defining a representation can also be side-stepped by using non-parametric methods (Ormoneit and Sen, 2002; Rasmussen and Kuss, 2003), that implicitly use very rich representations that can adapt to the complexity of the data. Such methods have the disadvantage that they usually rely on inverting matrices that grow with the dataset, however. As a solution, efficient approximations can be used (Seeger et al., 2003; Snelson and Ghahramani, 2006; Rahimi and Recht, 2007).

We will discuss these issues in more detail in the remainder of this section. First, we will discuss various RL methods that provide stable policy updates. Subsequently, we will give an overview of different studies addressing RL with high-dimensional states, followed by a discussion of non-parametric RL techniques. Finally, we discuss various methods for efficiently approximating non-parametric methods that allow such methods to be applied to large data sets.

4.1 Policy Updates with Limited Information Loss.

Perhaps the most straightforward way to limit the information loss of a policy is to stay close to a previous policy. Policy gradient methods (Williams, 1992; Sutton et al., 1999), for example, limit the policy update $\delta\boldsymbol{\theta}$ to a step in the gradient direction of fixed Euclidean length $\delta\boldsymbol{\theta}^T\delta\boldsymbol{\theta} = \epsilon$. However, this metric is not invariant to re-parametrization of the policy. This problem can be addressed by instead using the Fisher information metric $\mathbf{F}$ in the constraint $\delta\boldsymbol{\theta}^T\mathbf{F}(\boldsymbol{\theta})\delta\boldsymbol{\theta} = \varepsilon$, as suggested by Kakade (2002). This constraint can be interpreted as the second-order Taylor expansion of the loss of information (relative entropy) between the path distributions of the original and the updated policy (Bagnell and Schneider, 2003a).

There are two main approaches to specifying such an information constraint. As suggested by Bagnell and Schneider (2003a), the loss of information between successive path distributions might be bounded. Such a formulation is equivalent to bounding the expected loss of information between successive policies, since the transition dynamics are the same under both path distributions. In our previous work (Peters et al., 2010), we instead proposed to bound the information loss between successive state-action distributions.

The first formulation has led to various algorithms that provide stable policy updates. For example, the dynamic policy programming algorithm (Azar et al., 2011) uses the relative entropy between successive policies as an additional cost term in the value function. Similar update equations have been derived from two points of view. Firstly, from the point of view of maximizing the cumulative reward under constraints on the communication bandwidth or data processing capacity (Tishby and Polani, 2011), or, equivalently, minimizing the policy complexity under a constraint on the policies value (Still and Precup, 2012). Another derivation minimizes the relative entropy from the trajectory distribution conditioned on obtaining maximal rewards to the proposed policy (Rawlik et al., 2013b). In these approaches, the trade-off between greedy exploitation and maintaining stable updates—analogous to the (inverse) temperature of a Boltzmann distribution—is a free parameter. To obtain convergence, an update schedule that decays the temperature over time has to be designed. In contrast to adding the relative entropy as a cost-term, Levine and Abbeel (2014) employ a bound on the relative entropy between successive policies.

Another line of work, that is related to bounding the divergence between successive policies, has focused on guaranteeing improvement of the policy by limiting the policy update. In conservative policy iteration (Kakade and Langford, 2002) and safe policy iteration (Pirotta et al., 2013), the updated policy is a mixture of the old policy and a greedy policy that maximized a lower bound on the policy improvement. Trust region policy optimization (Schulman et al., 2015) similarly guarantees policy improvement, using the relative entropy between successive policies in the lower bound.

Kober et al. (2013) observed that “in practice, reinforcement learning algorithms tend to work best for real systems when they are constrained to make modest changes to the distribution over states while learning”. However, depending on the system dynamics, a small change in the policy might cause a large change in the distribution of states visited by the policy. Thus, it may be advantageous to limit the relative entropy between successive state-action distributions rather than between successive policies or trajectory distributions. Empirically, a bound on the state-action distribution has been shown to outperform a bound on the policy for relatively large step-sizes (Lioutikov et al., 2014). Moreover, bounding the information loss between subsequent state-action distributions has been shown to have optimal regret in an adversarial Markov decision process (MDP) settings for discrete finite horizon problems (Zimin and Neu, 2013).

In our previous work (Peters et al., 2010), the Relative Entropy Policy Search (REPS) algorithm was derived based on such a bound on the relative entropy between successive state-action distributions. Policies learned using REPS re-weight the previous state-action distribution using a soft-max of the advantage function. This soft-max policy is reminiscent of the common ad-hoc exploration-exploitation trade-off using Boltzmann exploration (Kaelbling et al., 1996; Sutton, 1990; Lin, 1993), expectation-maximization based updates (Peters and Schaal, 2007; Kober et al., 2011), and the previously discussed algorithms by Azar et al. (2011) and Rawlik et al. (2013b). The advantage of REPS is, that the ‘temperature’ parameter that governs exploration is set directly by the algorithm. As a result, the algorithm is invariant to re-scaling of the reward function, and the temperature automatically decreases as the policy converges.

Alternative techniques search an optimal policy within the space of policies of similar exponential form (Lever and Stafford, 2015; Bagnell and Schneider, 2003b). In REPS, the exponential form does not result from an imposed search-space of the policy, but is a direct consequence of optimizing the bounded optimization problem. Under specific assumptions on the environment and the reward functions, exponential transformations of the value functions are also used in the definition of the optimal policy in the non-parametric method by Rawlik et al. (2013a). The embedding of the exponentiated value function, however, suffers from numerical problems where the value function is low.

Earlier approaches based on the REPS formulation (Lioutikov et al., 2014; Kupcsik et al., 2013; Peters et al., 2010; Daniel et al., 2016) have shown to be successful on a variety of problems. In all these approaches, a function-valued Lagrangian multiplier V emerges from the resulting optimization problem, which can be seen as a value function (Peters et al., 2010). However, these approaches assume the Lagrangian multiplier V is linear in manually-defined features, making it hard to apply the algorithm to domains with high-dimensional sensor representations. We relaxed this assumption, by requiring V to be a member of a non-linear RKHS, allowing implicit infinite feature representations. In contrast to earlier approaches based on the REPS optimization problem, our method can naturally handle non-parametric policies, such as Gaussian process policies.

REPS requires the estimation of the Bellman error. This estimation can for example be performed using a learned transition model. Thus far, work on learned transition models for REPS has been limited. The transition dynamics have been approximated by deterministic single-sample outcomes (Daniel et al., 2016; Peters et al., 2010) which only works well for deterministic environments, or by time-dependent linear models (Lioutikov et al., 2014). Gaussian process models have been used in the bandit setting (Kupcsik et al., 2013) to learn a simulator that predicts the outcome of new roll-outs. Instead, similar to previous value function methods (Grünewälder et al., 2012b; Boots et al., 2013; Nishiyama et al., 2012), we will employ empirical conditional RKHS embeddings to obtain non-linear models for step-based reinforcement learning.

4.2 Reinforcement Learning for High-dimensional State Representations.

Recently, several researchers have started to address the problem of reinforcement learning with high-dimensional state representations such as camera images. In many of these works, learning consists of two fully separate steps: learning a representation and, subsequently, learning a policy or value function. One possible approach is to view the problem as reducing the dimensionality of the high-dimensional states while losing as little information as possible. For such a purpose, deep neural networks such as deep auto-encoders have become popular. For example, Lange et al. (2012) and Mattner et al. (2012) used such networks to learn state representations that were subsequently exploited to learn visual tasks using fitted Q-iteration or non-parametric approximate dynamic programming, respectively. Finn et al. (2016) used spatial auto-encoders, that exploit the spatial coherence of images, to learn visual pushing tasks using guided policy search.

The discussed methods do not explicitly make use of the structure of reinforcement learning problems. In domains where salient sensory distractors occur, taking such structure into account can help avoid representing those distractors. Jonschkowski and Brock (2015) proposed a method that takes observed transitions and rewards into account using a set of robotic priors, to learn representations for, among others, a visual navigation task. Another way to take the dynamics into account is to use slow-feature analysis, which encodes diffusion distances based on the transition kernel (Böhmer et al., 2013). The learned representations were used in an LSPI approach to learn visual navigation. A visual navigation task was also addressed by Boots et al. (2011), who select features based on their ability to predict characteristics of possible future events. Forcing the dynamics to be linear in the latent space is another way of enforcing a task-relevant representation. Watter et al. (2015) showed this principle to be successful at learning representations for model-predictive control of several dynamical systems in an off-policy setting.

The goal of finding a state representation that takes the task structure into account can also be reached by directly learning neural network policies that map from high-dimensional sensory information to control actions (Levine et al., 2016). In this approach, the hidden neurons act as feature representation. As the network is trained to reproduce correct actions, those neurons are optimized to provide a representation that helps the learning agent succeed at its task. This method requires having a low-dimensional task representation available at train time. It guides the neural network training using locally optimized trajectories. Mnih et al. (2015) instead directly used a neural network as an approximate Q function, as such an approach is able to scale to large networks and datasets. Their method obtained superb performance on playing Atari games, but is limited to problems with a discrete set of actions. Other methods have used neural networks that represent policies and have achieved impressive results on simulated dynamical systems, including bipedal locomotion and a driving simulator (Schulman et al., 2015; Mnih et al., 2016; Lillicrap et al., 2016). These methods, however, tend to require datasets a million transitions or more to achieve these results. Such large datasets are impractical to obtain for real physical systems, e.g., robots. Furthermore, all neural-network based learning methods rely on non-convex optimization of all the network weights.

4.3 Non-parametric Reinforcement Learning Methods.

Non-parametric kernel methods use an implicit representation of the data, and therefore avoid the explicit choice of a feature representation. Many different approaches in reinforcement learning have been re-formulated to use non-parametric methods. For example, non-parametric value iteration methods have been proposed for (partially observable) MDPs (Grünewälder et al., 2012b; Nishiyama et al., 2012), and non-parametric approximate dynamic programming method have been proposed by Ormoneit and Sen (2002), Taylor and Parr (2009), Deisenroth et al. (2009), Kroemer and Peters (2011) and Xu et al. (2014). Engel et al. (2003) proposed a Gaussian process (GP) temporal difference algorithm, and furthermore, there are examples of non-parametric policy iteration schemes such as kernelized least-squares policy evaluation (Jung and Polani, 2007) and least-squares policy iteration (Xu et al., 2007; Rasmussen and Kuss, 2003). The approximate linear programming algorithm, proposed by Pazis and Parr (2011), does not use kernels. Instead, this model-free method assumes the value function is Lipschitz, and assumes deterministic dynamics. Such value function methods use greedy maximization with respect to approximated value functions. Consequentially, these methods use deterministic actions. If exploration of the state space is required, heuristics such as an ϵ-greedy or a soft-max policy can be used. Furthermore, Grünewälder et al. (2012b) and Nishiyama et al. (2012) consider only discrete action sets and assume the state-action space can be sampled uniformly. For robotic systems, this can generally only be done in simulation.

Policy-search methods can be applied to address these shortcomings, by iteratively improving a policy. Kober et al. (2011) introduced cost-regularized kernel regression, which finds non-parametric policies for contextual bandits. In contrast, other approaches have focused on step-based decision making. For example, Bagnell and Schneider (2003b) developed a policy gradient method embedding a desirability function that defines a policy in a RKHS. However, their approach is restricted to discrete actions, and as a model-free method, cannot exploit learnable system dynamics. More recently, Lever and Stafford (2015) introduced another policy gradient approach, which searches for the mean function of a Gaussian process policy within a RKHS. Although this method cannot represent non-Gaussian policies unlike the method of Bagnell and Schneider (2003b), it can represent policies that are close to deterministic more easily. A disadvantage of this method is that a schedule to decay the co-variance of the Gaussian towards zero has to be manually defined. Vien et al. (2016) introduced a non-parametric variant of the *natural* policy gradient, together with a natural actor-critic algorithm and expectation-maximization based updates.

Alternative non-parametric methods were proposed by Rawlik et al. (2013a) and Deisenroth and Rasmussen (2011). The method by Rawlik et al. (2013a) considers continuous-time systems with continuous actions. This method assumes the environments injects observable control noise and that the system is control-affine. Deisenroth and Rasmussen (2011) describe a model-based iterative method. They explicitly marginalize the uncertain non-parametric model to avoid over-greedy optimization. However, their method requires the reward function to be known and to be of squared exponential form. Additionally, it selects action greedily and so it does not address the exploration problem.

4.4 Efficient Approximation for Non-parametric RL Methods.

Kernel-based non-parametric methods usually requires inverting Gram matrices, which are of dimension $n \times n$, where n is the number of data points. This step is a bottleneck for scaling up these methods to large data sets (1000 - 10000 samples being the upper limit where most of these methods start to be prohibitively slow). Therefore, non-parametric approaches benefit from efficient approximations for large datasets.

Multiple non-parametric RL algorithms handle large datasets by selecting a subset of data points, and then finding coefficients for each of the kernels using a quadratic programming problem (Engel et al., 2003; Xu et al., 2014; Jung and Polani, 2007; Xu et al., 2007; Lever and Stafford, 2015). In most cases, this sparsification approach results in an approximation of the kernel function by a non-stationary kernel function parametrized by a subset of active data-points, as pointed out by Jung and Polani (2007) and Xu et al. (2007). If all data-points were chosen to be active, this approximation would be exact.

A sparsification approach for the method of Ormoneit and Sen (2002) has been proposed by Barreto et al. (2011). The proposed stochastic factorization approach, however, works only on stochastic kernel matrices, such as used in Nadaraya-Watson kernel regression. Another approach for applying cost-regularized kernel regression to large data sets is to apply a learning algorithm to separate subsets of the data and combine the results (Macedo et al., 2014). However, as the authors state, this approximation is not mathematically equivalent to the original problem, and if, as suggested, data from multiple iterations is combined, the on-policy assumption of the algorithm would be violated.

Yet another method takes an approach complimentary to sparsification approaches. Whereas sparse Gaussian process approaches essentially find coefficients for the true kernel matrix at a subset of data points, Rahimi and Recht (2007) propose an approach that evaluates an approximation to the kernel function at all training data points. To the best of our knowledge, this type of approximations has, so far, not been explored in the context of reinforcement learning. This approach approximates the kernel function as an inner product of random Fourier features. As such, it is reminiscent of the work by Fard et al. (2013) and Ghavamzadeh et al. (2010), who use random projections for parametric value function methods. However, these studies aim at reducing the dimensionality of sparse or redundant features using random projections, the method of Rahimi and Recht (2007) is used to project low-dimensional vectors into high-dimensional feature spaces so that the function approximation problems become linear. Konidaris et al. (2011) employ Fourier basis features for value function approximation. The parameters of the basis features were predetermined whereas in our approach the parameters are drawn from a distribution based on the kernel bandwidth, which can be optimized using standard techniques.

5. Conclusion and Future Work

In this paper, we have developed a policy search method with smooth, robust updates to solve continuous MDPs. Our method uses learned non-parametric models and allows the use of non-parametric policies, avoiding hand-crafted features. By taking the sampling distribution into account during policy updates, stable learning progress was obtained even with relatively small batches of data. By embedding the conditional transition distribution, expectations over functions of the next state can be computed without density estimation. The resulting predictions are robust even with state representations with hundreds of dimensions.

We show that the resulting algorithm is able to outperform other non-parametric algorithms on a reaching task and a pendulum swing-up task with control noise in on-policy settings. We also show good performance of the proposed algorithm relative to neural-network based methods on a variant of the puddle-world task with a redundant input representation.

A limiting factor in applying the method to larger problems is the computational cost of inverting the Gram matrix. To address this issue, we evaluated different approaches to approximate this matrix that allow the inverse to be calculated efficiently. We found the approximation of the kernel function using random Fourier features to have desirable properties: they are computationally fast, easy to implement, and yielded good performance for moderate numbers of basis features. Sparsification yielded better performance on a smaller set of basis functions.

We evaluated the applicability of the algorithm to a real-robot underpowered swing-up task, with 600-dimensional visual representations of the pendulum's state. Here, we found that our method could successfully learn policies that swing up and balance the pendulum, from high-dimensional data.

Many tasks concerning sensory data are similar to the real-robot underpowered swing-up task, in that they have a high extrinsic dimensionality, but are intrinsically low-dimensional. Kernel-based algorithms perform all operations on kernel values, so they are invariant to the extrinsic dimensionality. Our kernel-based RL algorithm can, thus, be applied to such tasks without a separate dimension reduction. However, underlying distractor dimensions would negatively impact kernel-based algorithms unless the kernel parameters would be set (by design or using learning techniques) to ignore those effects.

The proposed algorithm is designed with data scarcity in mind, and shows good empirical performance in such cases. However, even with the proposed approximation of the Gram matrix, the runtime of the algorithm can still be a limiting factor. As the complexity of learning problems grows, we need more data and more approximating features, increasing the computational requirements. Thus, the proposed algorithm is likely to be a good choice when obtaining samples is costly or restricted, but might not be the best choice when samples are cheap relative to computation time.

In future work, we want to investigate synergies between the optimization problem and the generalizing policy. We will also address hyper-parameter optimization in systems with multi-dimensional controls, as our current objective does not always yield desired results. We plan to apply our algorithm on real-world robotic tasks, including tasks with high-dimensional visual or tactile sensors, and investigate how to exploit structural knowledge about dynamical systems in such tasks. To avoid the need to enhance visual salience, learning task-relevant kernels will be essential.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreements #610967 (TACMAN) and #270327 (CompLACS). Additionally, this project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement #645582 (RoMaNS). Part of this research has been made possible by the provision of compute time on the Lichtenberg cluster of the TU Darmstadt.

References

M. G. Azar, V. Gómez, and B. Kappen. Dynamic policy programming with function approximation. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AIstats)*, pages 119–127, 2011.

J.A. Bagnell and J. Schneider. Covariant policy search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003a.

J.A. Bagnell and J. Schneider. Policy search in reproducing kernel Hilbert space. Technical Report RI-TR-03-45, CMU, 2003b.

A. S. Barreto, D. Precup, and J. Pineau. Reinforcement learning using kernel-based stochastic factorization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 720–728, 2011.

P. L. Bartlett. An introduction to reinforcement learning theory: Value function methods. In *Advanced Lectures on Machine Learning*, pages 184–202. Springer Berlin Heidelberg, 2003.

D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific Belmont, MA, 1995.

D. P. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

W. Böhmer, S. Grünewälder, Y. Shen, M. Musial, and K. Obermayer. Construction of approximation spaces for reinforcement learning. *The Journal of Machine Learning Research*, 14(1):2067–2118, 2013.

B. Boots, S. Siddiqi, and G. Gordon. Closing the learning planning loop with predictive state representations. *International Journal of Robotics Research*, 30:954–956, 2011.

B. Boots, A. Gretton, and G. J. Gordon. Hilbert space embeddings of predictive state representations. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013.

L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming using Function Approximators*, volume 39. CRC press, 2010.

C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 2016.

P. Dayan and G.E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

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 (ICML)*, pages 465–472, 2011.

M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7):1508–1524, 2009.

M. P. Deisenroth, G. Neumann, and J. Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, pages 388–403, 2013.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

Y. Engel, S. Mannor, and R. Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 20, pages 154–161, 2003.

M. M. Fard, Y. Grinberg, A.-M. Farahmand, J. Pineau, and D. Precup. Bellman error based feature generation using random projections on sparse spaces. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3030–3038, 2013.

C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *Proceedings of the IEEE International Conference on Robotics and Automations (ICRA)*, 2016.

A. Gelman, J.B. Carlin, H.S. Stern, and D.B. Rubin. *Bayesian Data Analysis.* Chapman & Hall/CRC, 2 edition, 2004.

M. Ghavamzadeh, A. Lazaric, O. Maillard, and R. Munos. LSTD with random projections. In *Advances in Neural Information Processing Systems (NIPS)*, pages 721–729, 2010.

S. Grünewälder, G. Lever, L. Baldassarre, S. Patterson, A. Gretton, and M. Pontil. Conditional mean embeddings as regressors. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1823–1830, 2012a.

S. Grünewälder, G. Lever, L. Baldassarre, M. Pontil, and A. Gretton. Modelling transition dynamics in MDPs with RKHS embeddings. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 535–542, 2012b.

J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 918–926, 2014.

T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, pages 1171–1220, 2008.

R. Jonschkowski and O. Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015.

T. Jung and D. Polani. Kernelizing LSPE(λ). In *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 338–345, 2007.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

S. A. Kakade. Natural policy gradient. In *Advanced in Neural Information Processing Systems (NIPS)*, 2002.

S. A. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 2, pages 267–274, 2002.

J. Kober, E. Oztop, and J. Peters. Reinforcement learning to adjust robot movements to new situations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2650–2655, 2011.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 11(32):1238–1274, 2013.

G. D. Konidaris, Osentoski S., and Thomas P. S. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 380–385, 2011.

O. Kroemer and J. Peters. A non-parametric approach to dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1719–1727, 2011.

A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann. Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2013.

M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, December 2003.

S. Lange, M. Riedmiller, and A. Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *International Joint Conference on Neural Networks*, 2012.

G. Lever and R. Stafford. Modelling policies in MDPs in reproducing kernel Hilbert space. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AIstats)*, pages 590–598, 2015.

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

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

L.-J. Lin. *Reinforcement Learning for Robots using Neural Networks*. PhD thesis, Carnegie Mellon University, 1993.

R. Lioutikov, A. Paraschos, J. Peters, and G. Neumann. Generalizing movements with information theoretic stochastic optimal control. (9):579–595, 2014.

Z. Lu, D. Guo, A. Bagheri Garakani, K. Liu, A. May, A. Bellet, L. Fan, M. Collins, B. Kingsbury, M. Picheny, and F. Sha. A comparison between deep neural nets and kernel acoustic models for speech recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 2016.

J. Macedo, C. Santos, and L. Costa. Using cost-regularized kernel regression with a high number of samples. In *Proceedings of the IEEE International Conference on Autonomous Robot Systems and Competitions*, pages 199–204, 2014.

S. Mannor, D. Simester, P. Sun, and J. N. Tsitsiklis. Biases and variance in value function estimates. *Management Science*, 53(2):308–322, February 2007.

J. Mattner, S. Lange, and M. Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In *International Conference on Neural Information Processing*, pages 126–133, 2012.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.

Y. Nishiyama, A. Boularias, A. Gretton, and K. Fukumizu. Hilbert space embeddings of POMDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 644–653, 2012.

D. Ormoneit and Ś. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2-3): 161–178, 2002.

R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 752–759, 2008.

J. Pazis and R. Parr. Non-parametric approximate linear programming for MDPs. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 459–464, 2011.

J. Peters and S. Schaal. Policy gradient methods for robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2219–2225, 2006.

J. Peters and S. Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 745–750, 2007.

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

J. Peters, J. Kober, and D. Nguyen-Tuong. Policy learning – a unified perspective with applications in robotics. In *Proceedings of the European Workshop on Reinforcement Learning (EWRL)*, 2008.

J. Peters, K. Mülling, and Y. Altün. Relative entropy policy search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI), Physically Grounded AI Track*, pages 1607–1612, 2010.

M. Pirotta, M. Restelli, A. Pecorino, and D. Calandriello. Safe policy iteration. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 307–315, 2013.

W. B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1177–1184, 2007.

C. E. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, volume 4, page 1, 2003.

K. Rawlik, M. Toussaint, and S. Vijayakumar. Path integral control by reproducing kernel Hilbert space embedding. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013a.

K. Rawlik, M. Toussaint, and S. Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3052–3056. AAAI Press, 2013b.

B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. J. Smola. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, 1999.

B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In David Helmbold and Bob Williamson, editors, *Computational Learning Theory*, volume 2111 of *Lecture Notes in Computer Science*, pages 416–426. Springer Berlin Heidelberg, 2001.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

M. Seeger, C.K.I. Williams, and N.D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Workshop on Artificial Intelligence and Statistics*, 2003.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1257–1264, 2006.

L. Song, J. Huang, A. Smola, and K. Fukumizu. Hilbert space embeddings of conditional distributions with applications to dynamical systems. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 961–968, 2009.

L. Song, K. Fukumizu, and A. Gretton. Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models. *Signal Processing Magazine, IEEE*, 30(4):98–111, 2013.

S. Still and D. Precup. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 131(3):139–148, 2012.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 216–224, 1990.

R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1038–1044, 1996.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* MIT press, 1998.

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1057–1063, 1999.

C. Szepesvári. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010.

G. Taylor and R. Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1017–1024, 2009.

N. Tishby and D. Polani. Information theory of decisions and actions. In *Perception-Action Cycle*, pages 601–636. Springer, 2011.

H. van Hoof, T. Hermans, G. Neumann, and J. Peters. Learning robot in-hand manipulation with tactile features. In *Proceedings of the IEEE International Conference on Humanoid Robots (Humanoids)*, pages 121–127, 2015a.

H. van Hoof, J. Peters, and G. Neumann. Learning of non-parametric control policies with high-dimensional state features. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AIstats)*, pages 995–1003, 2015b.

N.A. Vien, P. Englert, and M. Toussaint. Policy search in reproducing kernel Hilbert space. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

M. Watter, J. T. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2728–2736, 2015.

M. Wiering and M. Otterlo. *Reinforcement learning: State-of-the-art.* Springer Berline Heidelberg, 2012.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

X. Xu, D. Hu, and X. Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007.

X. Xu, C. Lian, L. Zuo, and H. He. Kernel-based approximate dynamic programming for real-time online learning control: An experimental study. *IEEE Transactions on Control Systems Technology*, 22(1):146–156, 2014.

A. Zimin and G. Neu. Online learning in episodic Markovian decision processes by relative entropy policy search. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1583–1591, 2013.

Appendix A. The Dual and its Derivatives

To obtain the dual function, we re-insert the state-action probabilities $p_\pi = \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})$ in the Lagrangian to obtain the dual

$$\begin{aligned} g(\eta, V, \lambda) =& \lambda + \eta\epsilon + \mathbb{E}_{p_\pi(\mathbf{s},\mathbf{a})}\left[\delta(\mathbf{s},\mathbf{a},V) - \lambda - \eta \log \frac{p_\pi(\mathbf{s},\mathbf{a})}{q(\mathbf{s},\mathbf{a})}\right], \\ =& \lambda + \eta\epsilon + \mathbb{E}_{p_\pi(\mathbf{s},\mathbf{a})}\left[-\lambda + \lambda\right] + \mathbb{E}_{p_\pi(\mathbf{s},\mathbf{a})}\left[\delta(\mathbf{s},\mathbf{a},V) - \delta(\mathbf{s},\mathbf{a},V) + \eta\right], \\ =& \lambda + \eta\epsilon + \mathbb{E}_{p_\pi(\mathbf{s},\mathbf{a})}\eta \; \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s} = \lambda + \eta\epsilon + \eta = \eta\epsilon + \eta\log(Z), \\ =& \eta\epsilon + \eta\log\left(\mathbb{E}_{q(\mathbf{s},\mathbf{a})}\exp\left(\delta(\mathbf{s},\mathbf{a},V)/\eta\right)\right), \end{aligned}$$

where we used that $\exp(-\lambda/\eta - 1) = Z^{-1}$, so $\lambda + \eta = \eta\log(Z)$. The expected value over q can straightforwardly be approximated by taking the average of samples $1, \ldots, n$ taken from q. Note that λ and q do not appear in the final expression.

$$g(\eta, V) = \eta\epsilon + \eta\log\left(\frac{1}{n}\sum_{i=1}^{n}\exp\left(\delta(\mathbf{s}_i,\mathbf{a}_i,V)/\eta\right)\right).$$

When employing the kernel embedding, the Bellman error is written as

$$\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha}) = \mathcal{R}_{\mathbf{s}_i}^{\mathbf{a}_i} + \boldsymbol{\alpha}^T(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_{\mathrm{s}}(\mathbf{s}_i)).$$

We define

$$w_i = \frac{\exp\left(\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})/\eta\right)}{\sum_{i=j}^{n}\exp\left(\delta(\mathbf{s}_j,\mathbf{a}_j,\boldsymbol{\alpha})/\eta\right)}$$

to keep equations brief and readable. The partial derivatives can be written as:

$$\begin{aligned} \frac{\partial g(\eta,\boldsymbol{\alpha})}{\partial\eta} =& -\frac{1}{\eta}\sum_{i=1}^{n} w_i\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha}) + \epsilon + \log\left(\frac{1}{n}\sum_{i=1}^{n}\exp\left(\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})/\eta\right)\right), \\ \frac{\partial g(\eta,\boldsymbol{\alpha})}{\partial\boldsymbol{\alpha}} =& \sum_{i=1}^{n} w_i\left(\mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_{\mathrm{s}}(\mathbf{s}_i)\right), \end{aligned}$$

and furthermore, we obtain the Hessian:

$$\begin{aligned} \mathbf{H}(g(\eta,\boldsymbol{\theta}) =& \frac{1}{\eta}\sum_{i=1}^{n}\left(w_i\begin{bmatrix} -\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha}) \\ \mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_{\mathrm{s}}(\mathbf{s}_i)\end{bmatrix}\begin{bmatrix} -\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha}) \\ \mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_{\mathrm{s}}(\mathbf{s}_i)\end{bmatrix}^T\right) \\ & -\frac{1}{\eta}\sum_{i=1}^{n}\left(w_i\begin{bmatrix} -\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha}) \\ \mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_{\mathrm{s}}(\mathbf{s}_i)\end{bmatrix}\right)\sum_{i=1}^{n}\left(w_i\begin{bmatrix} -\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha}) \\ \mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_{\mathrm{s}}(\mathbf{s}_i)\end{bmatrix}\right)^T. \end{aligned}$$

The Hessian of a function is positive semidefinite on a set if and only if the function is convex on that set. Since η is the Lagrange multiplier for an inequality constraint, we have $\eta \geq 0$.To show that $\mathbf{H}(g(\eta,\boldsymbol{\theta}))$ is positive semidefinite on $(\eta,\theta) \in [0,+\infty) \times \mathbb{R}^n$ we introduce the following abbreviation:

$$\mathbf{u}_i = \begin{bmatrix} -\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha}) \\ \mathbf{K}\beta(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{k}_{\mathrm{s}}(\mathbf{s}_i)\end{bmatrix}^T.$$

Now, we can re-write the Hessian

$$\begin{aligned}\mathbf{H}(g(\eta,\boldsymbol{\alpha})) =& \frac{1}{\eta}\sum_{i=1}^{n}\left(w_i\mathbf{u}_i\mathbf{u}_i^T\right) - \frac{1}{\eta}\sum_{i=1}^{n}(w_i\mathbf{u}_i)\sum_{i=1}^{n}(w_i\mathbf{u}_i)^T\\ =& \frac{1}{\eta}\sum_{i=1}^{n}\left(w_i\mathbf{u}_i\mathbf{u}_i^T\right) - \frac{2}{\eta}\sum_{i=1}^{n}(w_i\mathbf{u}_i)\sum_{j=1}^{n}(w_j\mathbf{u}_j)^T\\ &+\frac{1}{\eta}\sum_{i=1}^{n}(w_i)\sum_{j=1}^{n}(w_j\mathbf{u}_j)\sum_{j=1}^{n}(w_j\mathbf{u}_j)^T\\ =& \frac{1}{\eta}\sum_{i=1}^{n}w_i\left(\mathbf{u}_i\mathbf{u}_i^T - 2\mathbf{u}_i\sum_{j=1}^{n}(w_j\mathbf{u}_j)^T + \sum_{j=1}^{n}(w_j\mathbf{u}_j)\sum_{j=1}^{n}(w_j\mathbf{u}_j)^T\right)\eta\\ =& \frac{1}{\eta}\sum_{i=1}^{n}\left(w_i\left(\mathbf{u}_i-\sum_{j=1}^{n}w_j\mathbf{u}_j\right)\left(\mathbf{u}_i-\sum_{j=1}^{n}w_j\mathbf{u}_j\right)^T\right).\end{aligned}$$

Note that we have used the property that $\sum_{i=1}^{n} w_i = 0$. $\mathbf{H}$ is positive semidefinite if $\mathbf{a}^T\mathbf{H}\mathbf{a} \geq 0$ for any $\mathbf{a}$.

$$\begin{aligned}\mathbf{a}^T\mathbf{H}\mathbf{a} =& \mathbf{a}^T\left(\frac{1}{\eta}\sum_{i=1}^{n}\left(w_i\left(\mathbf{u}_i-\sum_{j=1}^{n}w_j\mathbf{u}_j\right)\left(\mathbf{u}_i-\sum_{j=1}^{n}w_j\mathbf{u}_j\right)^T\right)\right)\mathbf{a}\\ =& \frac{1}{\eta}\sum_{i=1}^{n}\left(w_i\mathbf{a}^T\left(\mathbf{u}_i-\sum_{j=1}^{n}w_j\mathbf{u}_j\right)\left(\mathbf{u}_i-\sum_{j=1}^{n}w_j\mathbf{u}_j\right)^T\mathbf{a}\right)\\ =& \frac{1}{\eta}\sum_{i=1}^{n}\left(w_i\left(\mathbf{a}^T\mathbf{u}_i-\sum_{j=1}^{n}w_j\mathbf{a}^T\mathbf{u}_j\right)^2\right).\end{aligned}$$

The factors $\left(\mathbf{a}^T\mathbf{u}_i - \sum_{j=1}^{n} w_j\mathbf{a}^T\mathbf{u}_j\right)^2$ are squares of real numbers and thus non-negative. Since the w_i are the result of a soft-max operation and thus non-negative, this holds for the weighted sum, too. Thus, for any $\eta \geq 0$, $\mathbf{a}^T\mathbf{H}\mathbf{a} \geq 0$ and g is convex.

Appendix B. Optimization with Respect to V

We want to show that at least one of the functions V minimizing the dual functional g can be represented using a weighted sum of kernel functions centered at the sample states, i.e., that

$$V^* = \sum_{\tilde{\mathbf{s}} \in \tilde{\mathcal{S}}} \alpha_{\tilde{\mathbf{s}}} k_{\mathrm{s}}(\tilde{\mathbf{s}}, \cdot), \tag{18}$$

where $\tilde{\mathcal{S}}$ is the set of states samples during the roll-outs. We follow some steps in the proof of Schölkopf et al. (2001). They consider arbitrary objective functions c mapping to $\mathbb{R} \cup \{\infty\}$ of the form

$$c((\mathbf{s}_1, y_1, V(\mathbf{s}_1)), \ldots, (\mathbf{s}_m, y_m, V(\mathbf{s}_m))), \tag{19}$$

which can represent an error function between the value of a function $V(\mathbf{s})$ at the samples $\mathbf{s}_i$ and the corresponding desired outputs y_i. In our case, we do not have desired output values y_i for our objective function. This is inconsequential as c can be arbitrary, and so can be independent of all y values.

Any function V can be written as $V = \sum_{\tilde{\mathbf{s}} \in \tilde{\mathcal{S}}} \alpha_{\tilde{\mathbf{s}}} k_{\mathrm{s}}(\tilde{\mathbf{s}}, \cdot) + v(\mathbf{s})$, where $v(\mathbf{s})$ is an additional bias term. If V is constrained to be in the Hilbert space defined by k, Schölkopf et al. [2001] show that c is independent of the bias term $v(\mathbf{s})$. This means that for any optimal V' that is not of the proposed form, there is a V^* of the proposed form that has the same objective value which is obtained by subtracting $v(\mathbf{s})$ from V'.

As the dual function g satisfies the conditions to cost function c, for us this means that there is at least one V^* optimizing g of the proposed form. Note that it is inconsequential that the dual g also depends on Lagrangian parameter η. For any optimum $(\eta^*, V^{*\prime})$, if $V^{*\prime}$ is not of the proposed form, projecting $V^{*\prime}$ on the proposed basis yields another function V^* that satisfies $g(\eta^*, V'^*) = g(\eta^*, V^*)$, so (η^*, V^*) must be an optimum as well.

Therefore, there is always at least one minimum of any such function c of the proposed form.

Appendix C. Feedback signals to avoid joint angle and velocity limits

In our real robot set-up, the control actions selected by the algorithm is the increment in the torque to be applied (proportional to the jerk). However, to avoid running into joint angle and velocity limits, the resulting torque is modified when these limits are approaches. Therefore, the actually applied torque at time step t

$$\tau^{(t+1)} = \max\left(\min\left(\tau_t + u + \tau_{\mathrm{s}}^{(t)} + \tau_{\mathrm{d}}^{(t)}, \tau_{\max}^{(t)}\right), \tau_{\min}^{(t)}\right),$$

where $\tau_{\mathrm{s}}^{(t)}$ and $\tau_{\mathrm{s}}^{(d)}$ are spring- and damper related terms that apply when the robot gets close to the joint limit, and $\tau_{\max}^{(t)}$ and $\tau_{\min}^{(t)}$ are the maximum and minimum torque, respectively, that can be applied without breaking the velocity limit. The definition of θ, as well as the joint limit and the area where the feedback terms are applied, are illustrated in Figure 6b.

The feedback terms are defined as follows. The spring-like term

$$\tau_{\mathrm{s}}^{(t)} = \begin{cases} 0 & \text{if } -5/4\pi < \theta < 1/4, \pi \\ 15(-5/4\pi - \theta) & \text{if } \theta < -5/4\pi, \\ 15(1/4\pi - \theta) & \text{if } 1/4\pi < \theta, \end{cases}$$

is applied whenever the joint gets close to the joint limit at $1/2\pi = -3/2\pi$. The damper-like term

$$\tau_{\mathrm{d}}^{(t)} = \begin{cases} 0.3(-5/4\pi - \theta)\dot{\theta} & \text{if } \theta < -5/4\pi \text{ and } \dot{\theta} > 0, \\ 0.3(1/4\pi - \theta)\dot{\theta} & \text{if } 1/4\pi < \theta \text{ and } \dot{\theta} < 0, \\ 0 & \text{otherwise,} \end{cases}$$

is applied in the same region, as the feedback has to be higher when the pendulum has a high velocity. To prevent the velocity from exceeding the velocity limit, additionally, the minimum and maximum torques

$$\tau_{\max}^{(t)} = \min\left(\tau_{\max}^{(t-1)} + 0.05, 20(3.85 - \dot{\theta}) + 4.5\cos(\theta)\right),$$

$$\tau_{\min}^{(t)} = \max\left(\tau_{\min}^{(t-1)} - 0.05, 20(-3.7 - \dot{\theta}) + 4.5\cos(\theta)\right),$$

are applied. The cosine term is a rough compensation for the torque induced by gravity. The term linear in $\dot{\theta}$ is a linear damping term. Furthermore, the system has some delays which tended to induce oscillations close to the maximum torque. Therefore, the maximum (respectively minimum) can only be increased (decreased) by a small amount relative to the previous value.