Numerical Quadrature for Probabilistic Policy Search

Julia Vinogradska, Bastian Bischoff, Jan Achterhold, Torsten Koller and Jan Peters

Abstract—Learning control policies has become an appealing alternative to the derivation of control laws based on classic control theory. Model-based approaches have proven an outstanding data efficiency, especially when combined with probabilistic models to eliminate model bias. However, a major difficulty for these methods is that multi-step-ahead predictions typically become intractable for larger planning horizons and can only poorly be approximated. In this paper, we propose the use of numerical quadrature to overcome this drawback and provide significantly more accurate multi-step-ahead predictions. As a result, our approach increases data efficiency and enhances the quality of learned policies. Furthermore, policy learning is not restricted to optimizing locally around one trajectory, as numerical quadrature provides a principled approach to extend optimization to all trajectories starting in a specified starting state region. Thus, manual effort, such as choosing informative starting points for simultaneous policy optimization, is significantly decreased. Furthermore, learning is highly robust to the choice of initial policy and, thus, interaction time with the system is minimized. Empirical evaluations on simulated benchmark problems show the efficiency of the proposed approach and support our theoretical results.

Index Terms—Policy Search, Control, Gaussian Processes, Reinforcement Learning

1 Introduction

LEARNING control has become a viable approach in both the machine learning and control community. Many successful applications impressively demonstrate the advantages of learning control [1], [2], [3], [4], [5], [6], [7], [8]. In contrast to classical control methods, learning control does not presuppose a detailed understanding of the underlying dynamics but tries to infer the required information from data. Thus, relatively little expert knowledge about the system dynamics is required and fewer assumptions, such as a parametric form and parameter estimates, must be made.

For real-world applications of learning control, it is desirable to minimize the system interaction time. Thus, approaches that explicitly learn a dynamics model are often preferred, as model-free methods can require a prohibitive amount of system interactions [9], [10], [11], [12]. However, one drawback of model-based methods is that modeling errors can derail learning, as the inherently approximate and frequently highly erroneous model is implicitly assumed to approximate the real dynamics sufficiently well [13]. Thus, solutions to the approximate control problem might result in policies that do not solve the control task for the true system dynamics. This model bias can have severe consequences especially when few data is available and, thus, the model predictions are highly uncertain. Hence, employing Gaussian processes (GPs) as forward models for learning control is particularly appealing as they incorporate uncertainty about the system dynamics estimate. GPs infer a distribution over all plausible models given the observed data instead of compromising on an approximate model and, thus, avoid severe modeling errors. Another major advantage of Gaussian process forward models is that stability analyses for such closed-loop control systems are available [14], [15], including automatic tools [15] which do not require any expert knowledge.

One difficulty of learning control based on a GP forward model is that the state distribution when applying a policy for several discrete time steps is analytically intractable. Thus, to evaluate a policy, approximations for multi-step-ahead predictions are required. Common choices include Monte Carlo sampling [16], linearization of the posterior mean [17] and moment matching [18]. While Monte Carlo methods suffer from slow convergence [19] and introduce the need for small learning rates due to noisy numerical gradients, the deterministic linearization and moment matching methods provide analytical gradients of the multi-step-ahead predictions. However, such methods approximate the state distribution as a Gaussian. As a consequence their expressivity is limited, e.g., moment matching and linearization cannot handle state distributions with multiple modes. Thus, such approximations suffer from severe inaccuracies, especially when the state distribution differs significantly from a Gaussian.

Another major drawback of Gaussian approximations for multi-step-ahead predictions is that learning is limited to optimizing locally around one trajectory. This problem arises from the limited expressivity of Gaussians. When a distribution with high variance is propagated through nonlinear dynamics, the predictive distribution is typically highly complex and cannot be represented sufficiently well by a Gaussian. Thus, approximating predictions at uncertain inputs by a Gaussian is feasible only for state distributions with low variance. As a workaround [20], multiple starting

- *J. Vinogradska is with the Bosch Center for Artificial Intelligence and the Intelligent Autonomous Systems Lab, TU Darmstadt.*
  E-mail: julia.vinogradska@de.bosch.com
- *J. Achterhold, T. Koller and B. Bischoff are with the Bosch Center for Artificial Intelligence.*
- *J. Peters is with the Intelligent Autonomous Systems Lab, TU Darmstadt and the Max-Planck Institute for Intelligent Systems.*

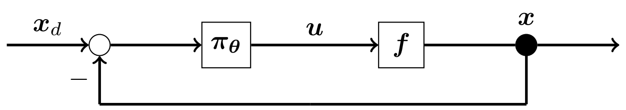


Fig. 1. A closed-loop control structure with controller $\boldsymbol{\pi}_{\boldsymbol{\theta}}$, system dynamics $\boldsymbol{f}$ and target state $\boldsymbol{x}^d$. We model the system dynamics $\boldsymbol{f}$ as a Gaussian process and assume that the policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ is parameterized by $\boldsymbol{\theta}$. The proposed algorithm learns $\boldsymbol{f}$ and $\boldsymbol{\theta}$ from interactions with the real system.

points that are representative of all starting states must be hand-selected to learn a suitable policy. The trajectories starting at the selected points must then be optimized simultaneously. As a result, the learned policy is highly dependent on the chosen starting points, suffers from bad generalization between those points and optimization is prone to local optima.

An alternative method to approximate multi-step-ahead predictions [15] employs numerical quadrature and provides analytical expressions for the state distribution. It approximates even complex distributions with multiple modes accurately and, additionally, bounds for the approximation error can be obtained. However, no policy learning was considered in [15]. A comparison of numerical quadrature and moment matching for long-term predictions is shown in Figure 2.

In this paper, we propose nuQuPS, a model-based policy search method that learns a Gaussian process forward dynamics model and relies on numerical quadrature to approximate multi-step-ahead predictions as introduced in [15]. The use of numerical quadrature increases data-efficiency and improves the quality of the learned policies significantly. Furthermore, numerical quadrature enables the use of arbitrary distributions for the starting state. Thus, it is possible to learn a policy, e.g., for all starting points in a certain region by choosing a uniform starting state distribution. Overall, the proposed algorithm is capable of learning global policies with remarkable data-efficiency, while no manual effort or expert knowledge are required.

The paper will be organized as follows: first, we specify the considered problem. In Section 1.2, we briefly review related work. Section 2 introduces the proposed algorithm, which is evaluated on multiple benchmark tasks in Section 3. A conclusion summarizes and discusses the provided results (Section 4).

1.1 Problem Statement

In this paper, we aim to learn to control a previously unknown dynamics system. We consider discrete-time dynamics

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\varepsilon} \tag{1}$$

with $\boldsymbol{x}_t, \boldsymbol{x}_{t+1} \in \mathbb{R}^D$, $\boldsymbol{u}_t \in \mathbb{R}^F$, unknown $\boldsymbol{f}$ and i.i.d. Gaussian measurement noise $\boldsymbol{\varepsilon} \in \mathcal{N}(0, \Sigma_{\boldsymbol{\varepsilon}})$. Figure 1 shows such a closed-loop control setting. Given a reward function $r\colon \boldsymbol{x}_t \mapsto r(\boldsymbol{x}_t) \in \mathbb{R}$, the goal is to find a policy $\boldsymbol{\pi}\colon \boldsymbol{x} \mapsto \boldsymbol{\pi}(\boldsymbol{x})$ that maximizes the expected reward up to time horizon T, when choosing $\boldsymbol{u}_t := \boldsymbol{\pi}(\boldsymbol{x}_t)$ for $t = 1, \ldots, T$. In policy search, we assume that the policy is parameterized by the parameter vector $\boldsymbol{\theta}$ and write $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x}_t) := \boldsymbol{\pi}(\boldsymbol{\theta}, \boldsymbol{x})$. The objective is then to find a parameter vector $\boldsymbol{\theta}^*$ that maximizes the expected long-term reward

$$R_{\boldsymbol{\pi}}(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}[r(\boldsymbol{x}_t)]. \tag{2}$$

We rely on Gaussian processes to model the system dynamics $\boldsymbol{f}$ from observations of the system behavior, as will be detailed in Section 2.1. To compute the expected long-term reward (2) for a particular policy, the state distributions $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$ must be determined. As with most nonlinear dynamics models, for our GP forward dynamics prediction of the next state becomes intractable when the input is a distribution. Thus, multi-step-ahead predictions must be approximated.

In our setting, it is desirable to learn a policy that is suitable not only for one starting state, but can control the system (1) reliably for a continuous region of starting states. More precisely, we aim to learn policies suitable for a given a starting state distribution $p(\boldsymbol{x}_0)$. We assume that $p(\boldsymbol{x}_0)$ is piecewise differentiable, but make no other assumptions e.g., that $p(\boldsymbol{x}_0)$ is Gaussian or has low variance. Thus, to approximate $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$ we propose the use of high performance numerical quadrature as in [15], see Section 2.4. To maximize the expected long-term reward (2), we perform gradient ascent. In Section 2.5 we will derive analytical expressions for $\partial R_{\boldsymbol{\pi}} / \partial \boldsymbol{\theta}$.

1.2 Related Work

The problem of deriving control laws when uncertainty is present has been considered in classic control theory for many years. In controller design, uncertainty may arise from modeling inaccuracies, the presence of (external) disturbances and the lack of data, as some system parameters are not known in advance but only during operation. Thus, a controller must be designed for a family of systems that is specified, e.g., by bounds for model parameters or for nonparametric uncertainties as bounds for the operator norm of the unknown dynamics. Robust control [21], [22] designs a single controller that is provably stable for all systems in the specified family – often at cost of overall controller performance. Adaptive control [23], [24] instead adjusts control parameters online in order to achieve prespecified performance, which can be computationally demanding. These methods rely on parametric dynamics models, which must be specified by an expert for each problem. In addition, both schemes require stability analysis of the dynamics system, e.g., via manually designed Lyapunov functions, which can be extremely challenging for complex, nonlinear systems.

Nonparametric system dynamics models are very appealing due to their high flexibility. They learn a dynamics model from data instead of relying on expert knowledge to pick a sufficiently accurate parametric form suitable for the dynamics. Nonparametric regression methods that learn the system dynamics from data have been considered in e.g., [1], [2], [7], [13], [25]. In [13], locally weighted Bayesian regression has been employed to model the system dynamics and uncertainty was treated as noise. To learn a policy, stochastic dynamic programming was applied on the discretized state space. The approaches [1], [2], [7], [25] model the forward dynamics as a Gaussian process.

Gaussian processes as forward models allow to incorporate uncertainty about the system dynamics without the need to discretize the state space. GPs have also been employed to model the system dynamics in [1], [25], [26], [27], [28], [29]. The approaches [26], [27], [28], [29] learn global value functions that are subsequently used to derive policies. For example, the PVI algorithm [29] learns the system dynamics and the value function, which are both modeled as GPs, in an episodic setting. While these value iteration based approaches provide great flexibility as no assumptions on the policy are made, maintaining a model for the global value function can be computationally demanding in large state spaces.

In contrast to such GP based fitted value iteration methods, PILCO [1], GPREPS [25] and [30] are policy search approaches that rely on GPs as forward dynamics models. The PILCO algorithm [1] employs GPs as forward dynamics models and conducts a search in the policy space, performing gradient ascent on the expected reward in an episodic setting. PILCO is particularly appealing, as analytical gradients of the expected reward with respect to the policy parameters are available. These analytical gradients enable scaling to high-dimensional problems and allow for highly flexible policies with many parameters.

The GP based value iteration approaches as well as the policy search PILCO and GPREPS approaches benefit from Bayesian averaging over all plausible models by incorporating the uncertainty provided by the GP dynamics models. However, propagating uncertainty through a Gaussian process is analytically intractable even for simple, Gaussian input distributions. All of the methods mentioned above rely on the moment matching approximation [18] or on sampling to propagate distributions through a GP. This moment matching approximation is only applicable to Gaussian input distributions and provides a good estimate of the next state distribution only if the variance of the input distribution is low. Unfortunately, these requirements typically do not hold during learning. In [31], the training of the GP dynamics is modified to improve the long-term state predictions obtained with cascaded moment matching, which leads to a significantly improved performance.

The sampling based approximations (Monte Carlo or kernel herding [32], [33], [34]) do not make use of the smoothness of a Gaussian process and, thus, require many samples to get a reliable estimate. Furthermore, the gradients of the expected long-term reward of a policy cannot be computed analytically for sampling methods. Thus, stochastic numerical gradients must be employed instead, which introduces noise and is costly if the number of parameters is high [1]. In [15], [35], an alternate approach to approximate the state distribution based on numerical quadrature was introduced. Numerical quadrature enables highly accurate approximations of the state distribution and can handle complex input distributions. However, no learning of policies was considered in [15] or [35]. In this paper, we employ the numerical quadrature approximation for multi-step-ahead predictions in policy search based on a GP dynamics model. Please note that our approach can easily be combined with a modified GP training as [31].

2 Numerical Quadrature Based Policy Search

We introduce nuQuPS, a model-based policy search approach with GPs as dynamics models and numerical quadrature for long-term predictions. First, we briefly recap Gaussian process regression which will be employed to model the system dynamics. Section 2.2 provides an overview of the nuQuPS algorithm. The choice of reward function and policy parametrization are discussed in Section 2.3. Subsequently, we elaborate on the proposed approximation for long-term predictions based on numerical quadrature. Finally, we compute analytic expressions for the gradients of the expected long-term reward with respect to the policy parameters $\boldsymbol{\theta}$ when numerical quadrature is used to propagate uncertainties.

2.1 Gaussian Process Regression

In the following, we will briefly recap Gaussian process regression as it will be used to learn the system dynamics from observed data.

Given noisy observations $\mathscr{D} = \{(\boldsymbol{z}^i, y^i = f(\boldsymbol{z}^i) + \varepsilon^i) \mid 1 \leq i \leq N\}$, where $\varepsilon^i \sim \mathcal{N}(0, \sigma_n^2)$, the prior on the values of f is $\mathcal{N}(0, K(Z,Z) + \sigma_n^2 I)$. The covariance matrix $K(Z,Z)$ is defined by the choice of covariance function k as $[K(Z,Z)]_{ij} = k(\boldsymbol{z}^i, \boldsymbol{z}^j)$. While the approach proposed in this paper is not limited to a certain kernel, we employ the squared exponential covariance function

$$k(\boldsymbol{z}, \boldsymbol{w}) = \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{w})^\intercal \Lambda^{-1}(\boldsymbol{z}-\boldsymbol{w})\right),$$

with signal variance σ_f^2 and squared lengthscales $\Lambda = \operatorname{diag}(l_1^2, \ldots, l_{D+F}^2)$ for all input dimensions. In general, any covariance function that is differentiable with respect to the inputs can be applied in the proposed approach.

Given a query point $\boldsymbol{z}_*$, the conditional probability of $f(\boldsymbol{z}_*)$ is

$$\begin{aligned} f(\boldsymbol{z}_*) \mid \mathscr{D} \sim \mathcal{N}\big(&\boldsymbol{k}(\boldsymbol{z}_*, Z)\boldsymbol{\beta}, \\ &k(\boldsymbol{z}_*, \boldsymbol{z}_*) - \boldsymbol{k}(\boldsymbol{z}_*, Z)(K(Z,Z) + \sigma_n^2 I)^{-1}\boldsymbol{k}(Z, \boldsymbol{z}_*)\big) \end{aligned} \quad (3)$$

with $\boldsymbol{\beta} = (K(Z,Z) + \sigma_n^2 I)^{-1}\boldsymbol{y}$. The hyperparameters, e.g., $\sigma_n^2, \sigma_f^2, \Lambda$ for the squared exponential kernel, are estimated by maximizing the log marginal likelihood of the data [36].

In this paper, we employ a Gaussian process $\boldsymbol{g}$ to model system dynamics. It takes state-action pairs $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{u})^\intercal$ and outputs differences to successor states, i.e., $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{g}(\boldsymbol{x}_t, \boldsymbol{u}_t)$. As these outputs are multivariate, we train conditionally independent GPs for each output dimension. We write $\sigma_{n,m}^2, \sigma_{f,m}^2, \Lambda_m$ for the GP hyperparameters in output dimension m and k_m for the corresponding covariance function.

2.2 Learning Policies from Scratch: nuQuPS

The proposed algorithm proceeds in an episodic setting. In the beginning, $\boldsymbol{\theta}$ is initialized randomly. The currently known best policy $\boldsymbol{\pi_\theta}$ is employed on the real system to get new information about the system dynamics. A starting point is sampled from the starting state distribution $p(\boldsymbol{x}_0)$ and a state-action trajectory $(\widetilde{\boldsymbol{x}}_0, \boldsymbol{\pi_\theta}(\widetilde{\boldsymbol{x}}_0)), \ldots, (\widetilde{\boldsymbol{x}}_{T-1}, \boldsymbol{\pi_\theta}(\widetilde{\boldsymbol{x}}_{T-1})), \widetilde{\boldsymbol{x}}_T$ is observed. This

Algorithm 1 Numerical Quadrature based Probabilistic Policy Search (nuQuPS)

```
Input: start distribution $p(\boldsymbol{x}_0)$, time horizon $T$, reward function $r$
Output: policy $\boldsymbol{\pi}_{\boldsymbol{\theta}^*}$ that maximizes $R_{\boldsymbol{\pi}}(\boldsymbol{\theta})$ (see Eq. (2))
 1: sample initial parameters $\boldsymbol{\theta} \sim \mathcal{N}(0, I)$
 2: $\mathscr{D} \leftarrow \emptyset$
 3: while not converged do
 4:     Sample $\widetilde{\boldsymbol{x}}_0 \sim p(\boldsymbol{x}_0)$
 5:     Compute rollout $\widetilde{\boldsymbol{x}}_0, \dots, \widetilde{\boldsymbol{x}}_T$ from $\widetilde{\boldsymbol{x}}_0$ employing $\boldsymbol{\pi}_{\boldsymbol{\theta}}$
 6:     $\mathscr{D}' \leftarrow \{(\widetilde{\boldsymbol{x}}_t, \boldsymbol{\pi}_{\boldsymbol{\theta}}(\widetilde{\boldsymbol{x}}_t), \widetilde{\boldsymbol{x}}_{t+1}) \mid t = 0, \dots, T-1\}$
 7:     $\mathscr{D} \leftarrow \mathscr{D} \cup \mathscr{D}'$
 8:     Train GP $\boldsymbol{g}$ on $\mathscr{D}$
 9:     Construct quadrature rule suited for $\boldsymbol{g}$ with Algorithm 2
10:     Compute $p(\boldsymbol{x}_1), \dots, p(\boldsymbol{x}_T)$ approximately with numerical quadrature (see Sec. 2.4)
11:     $R_{\boldsymbol{\pi}}(\boldsymbol{\theta}) \leftarrow \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}[r(\boldsymbol{x}_t)]$
12:     $\boldsymbol{\theta}^* \leftarrow \operatorname{argmax}_{\boldsymbol{\theta}} R_{\boldsymbol{\pi}}(\boldsymbol{\theta})$ via gradient ascent, $\partial R_{\boldsymbol{\pi}}(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ as in Sec. 2.5
13:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^*$
14: end while
15: return $\boldsymbol{\pi}_{\boldsymbol{\theta}^*}$
```

rollout data is used to update the GP dynamics model $\boldsymbol{g}$. With the updated dynamics model, the current policy is evaluated according to Equation (2). For this policy evaluation, GP predictions at the uncertain inputs $p(\boldsymbol{x}_0), \dots, p(\boldsymbol{x}_{T-1})$ must be computed. We employ numerical quadrature as described in Section 2.4 to propagate uncertainties through the GP. To ensure highly accurate approximate predictions and maintain computational efficiency in high dimensional state spaces, we tailor the quadrature rule to the GP dynamics $\boldsymbol{g}$ as described in Section 2.6 and Algorithm 2. To improve the policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$, the parameters $\boldsymbol{\theta}$ are updated to maximize $R_{\boldsymbol{\pi}}(\boldsymbol{\theta})$ via gradient ascent. Closed-form expressions for $\partial R_{\boldsymbol{\pi}}/\partial\boldsymbol{\theta}$ are given in Section 2.5. The current parameters $\boldsymbol{\theta}$ are set to the obtained optimal parameters $\boldsymbol{\theta}^*$, which can then be used for a rollout in the next episode. Algorithm 1 summarizes this approach.

Please note the conceptual similarity to PILCO, whose high level algorithm steps coincide with nuQuPS. The main difference between the two approaches is the way multi-step-ahead predictions are handled. While PILCO approximates the system state as a Gaussian at any time step, we choose a more flexible and powerful approximation method. The numerical quadrature based approximation is beneficial in several ways: (i) the controller performance and data efficiency is greatly improved by the more accurate state approximations, (ii) learning is more robust to different initializations of the policy and optimization is less likely to get stuck in a local optimum, (iii) there are less restrictions, e.g., on the parametric form of the policy, needed to compute the policy gradients analytically and, most importantly, (iv) due to the greater expressivity of numerical quadrature, policy optimization is not limited to local optimization around one trajectory (i.e., from one starting state). Instead, the policy can be optimized for a continuous region of starting states.

2.3 Choice of Policy Parametrization and Reward Function

In the following, we will elaborate on the possible choices of reward function and parametric form for the policy. In both cases, the restrictions result from our goal to provide analytical gradients to speed up policy search and make it scalable to high-dimensional problems. Fortunately, these restrictions allow for fairly flexible choices for both the policy and the reward function.

The reward function is used to determine the expected long-term cost of a policy. To evaluate a policy, the expected reward for all state distributions $p(\boldsymbol{x}_1), \dots, p(\boldsymbol{x}_T)$ when following this policy must be computed. Thus, the reward function r must be chosen such that $\mathbb{E}_{\boldsymbol{x}_t}[r(\boldsymbol{x}_t)]$ is analytically tractable for all $t = 0, \dots, T$. As we will see in Section 2.4, the state distribution is approximated by a Gaussian mixture model when applying numerical quadrature to propagate uncertainties. Due to linearity of the expectation, we conclude that r must be chosen such that expectations with respect to Gaussian distributions are analytically tractable. This holds, e.g., for all polynomials, Gaussians or mixtures of Gaussians [37] (which are a universal function approximators on any compact set). A straightforward choice for the reward function is to penalize the distance of the state to the target state $\boldsymbol{x}^d$, e.g.,

$$r(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^d)^\intercal \Sigma_r (\boldsymbol{x} - \boldsymbol{x}^d)\right), \tag{4}$$

where Σ_r defines width and orientation of the reward function. This reward function depends only on the current system state (and not, e.g., on the control signal $\boldsymbol{u}$). Please note that there is no technical requirement for r to be a function of $\boldsymbol{x}$ only, dependencies on other variables, e.g., on the control $\boldsymbol{u}$, can be included as long as the expectation of r remains differentiable with respect to the policy parameters $\boldsymbol{\theta}$. Irrespective of the technical feasibility of more complex reward functions we argue that a reward that only depends on $\boldsymbol{x}$ is very well suited for approaches that assume no expert knowledge of the system.

For the policy, we assumed that it is parameterized by a parameter vector $\boldsymbol{\theta}$. To optimize the policy, we aim to perform a gradient ascent employing analytic gradients of the expected long-term reward with respect to $\boldsymbol{\theta}$. Thus, the parametric form of the policy must be chosen such that it allows for analytic gradient computation. Additionally, in many real-world applications, the magnitude of the control signal is limited, e.g., by a constant $\boldsymbol{u}_{\max} \geq \boldsymbol{\pi}(\boldsymbol{x}) \geq -\boldsymbol{u}_{\max}$. These constraints must be incorporated in the parametric form. In [1], it was proposed to apply a squashing function such as, e.g., $\sigma(\boldsymbol{x}) = \sin(\boldsymbol{x})$ or $\sigma(\boldsymbol{x}) = 9/8 \sin(\boldsymbol{x}) + 1/8 \sin(3\boldsymbol{x})$. Also, analytic gradients of the GP prediction with respect to $\boldsymbol{\theta}$ were provided for (i) squashed linear policies $\boldsymbol{\pi}(\boldsymbol{x}) = u_{\max}\sigma(M\boldsymbol{x} + b)$ and (ii) squashed RBF policies $\boldsymbol{\pi}(\boldsymbol{x}) = \sigma(k(C, \boldsymbol{x})^\intercal \boldsymbol{\beta})$. Following [1], we squash the policies to limit the magnitude. Applying numerical quadrature for multi-step-ahead predictions, we will derive analytic policy gradients based on the gradients given in [1]. Thus, nuQuPS can handle squashed linear and squashed RBF policies. Additionally, our numerical quadrature approach allows to use sums of squashed linear policies, which corresponds to a Fourier series expansion of an arbitrary, bounded function.

2.4 Numerical Quadrature for Uncertainty Propagation

To evaluate a policy, multi-step-ahead predictions for a given GP dynamics $\boldsymbol{g}$ must be computed, see Equation (2). Given a query point $(\boldsymbol{x}_t, \boldsymbol{u}_t)$, the GP predicts the next state $\boldsymbol{x}_{t+1}$ to be normally distributed as stated in Equation (3). When the input $\boldsymbol{x}_t$ is uncertain, the next state distribution is given as

$$p(\boldsymbol{x}_{t+1}) = \int p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t) p(\boldsymbol{x}_t) d\boldsymbol{x}_t. \tag{5}$$

In general, this distribution is not Gaussian even if $p(\boldsymbol{x}_t)$ is. Furthermore, for most choices of covariance function, as e.g., for the widely employed squared exponential, $p(\boldsymbol{x}_{t+1})$ is analytically intractable and must be approximated. Please note that we omitted $\boldsymbol{u}_t$ for notational convenience in Equation (5) as $\boldsymbol{x}_t = \boldsymbol{\pi}(\boldsymbol{x}_t)$.

When $p(\boldsymbol{x}_t)$ is Gaussian, the first two moments of $p(\boldsymbol{x}_{t+1})$ can be computed in closed-form, which gives rise to the moment matching approximation [18]. The predictive distribution $p(\boldsymbol{x}_{t+1})$ is then approximated by a Gaussian with the computed moments. However, this approximation has some major drawbacks. First, approximating the state distribution by a Gaussian can lead to severe inaccuracies when the true state distribution is more complex, e.g., has multiple modes. Unfortunately, this is often the case when the system dynamics is highly nonlinear. Second, the first two moments of the predictive distribution can only be computed analytically when the input distribution is Gaussian. Starting with a Gaussian state distribution $p(\boldsymbol{x}_0)$, the state distribution $p(\boldsymbol{x}_1)$ will be approximated with a Gaussian. To compute $p(\boldsymbol{x}_2)$, the approximation of $p(\boldsymbol{x}_1)$ will be used as the model input. However, as the true distribution $p(\boldsymbol{x}_1)$ is not Gaussian, the moment matching approximation will not capture the first two moments of $p(\boldsymbol{x}_2)$ correctly. As a result, when cascading multiple moment matching steps, the computed moments do not match the true moments of the state distribution after only two steps. Typically, the computed moments will drift further away from the true moments with every time step. Figure 2 (cf. [15]) illustrates the described problems, that can occur when applying moment matching for long-term predictions.

As an alternative to moment matching, in [15] numerical quadrature was proposed to approximate the GP predictive distribution. Numerical quadrature approximates the value of an integral

$$\int_a^b f(\boldsymbol{x}) d\boldsymbol{x} \approx \sum_{i=1}^{p} w_i f(\boldsymbol{\xi}_i)$$

given a finite number p of function evaluations. A widely used class of quadrature rules are interpolatory quadrature rules, which integrate all polynomials up to a certain degree exactly. In this paper, we employ Gaussian quadrature rules, where the evaluation points $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_p$ are chosen to be the roots of certain polynomials from orthogonal polynomial families. They achieve the highest accuracy possible for univariate interpolatory formulæ [38]. For multivariate integrals, the quadrature problem is significantly harder. While many formulæ for the univariate case can straightforwardly be generalized to multivariate integrals, they often suffer from the curse of dimensionality. However, quadrature methods that scale better and are feasible for up to 20 dimensions have been developed. See [39] for an overview. In [15], it has been shown that numerical quadrature accurately approximates the state distribution and can be computed efficiently, see also Figure 2. However, no policy learning was considered in [15]. We follow this approach and approximate the integral from Equation (5) with numerical quadrature.

When learning to control a physical system, the state space is bounded in most cases (e.g., for temperatures, pressures, angles, lengths). Thus, we assume $\boldsymbol{x} \in X = [a_1, b_1] \times \cdots \times [a_D, b_D]$. Please note that this assumption is technically not necessary, as there exist quadrature rules that are suited for the domain $\mathbb{R}^D$ when integrating against a Gaussian weight function as in our case. However, we found the proposed quadrature rules, that are tailored to the GP dynamics, to perform significantly better than these generic rules. Given the state distribution $p(\boldsymbol{x}_t)$, the next state is determined by

$$p(\boldsymbol{x}_{t+1}) = \int_X p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t) p(\boldsymbol{x}_t) d\boldsymbol{x}_t. \tag{6}$$

We apply numerical quadrature to approximate this integral. We choose a composed Gaussian product quadrature rule, which will be detailed in Section 2.6. For now, it is sufficient to note that our quadrature rule provides a set of evaluation points $\mathbb{X}$ and positive weights w_n for all nodes $\boldsymbol{\xi}^n \in \mathbb{X}$. Integral (6) is then approximated by

$$p(\boldsymbol{x}_{t+1}) \approx \sum_{\boldsymbol{\xi}^n \in \mathbb{X}} w_n p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t = \boldsymbol{\xi}^n) p(\boldsymbol{x}_t = \boldsymbol{\xi}^n), \tag{7}$$

which results in a weighted sum of Gaussian distributions. The approximate state distribution at time $t+1$ can be written

$$p(\boldsymbol{x}_{t+1}) \approx \boldsymbol{\phi}^{\mathsf{T}} \boldsymbol{\alpha}_{t+1} \tag{8}$$

with $\alpha_{t+1,n} := w_n p(\boldsymbol{x}_t = \boldsymbol{\xi}^n)$ and $\phi_n(\boldsymbol{x}) := p(\boldsymbol{x}_{t+1} = \boldsymbol{x} \mid \boldsymbol{x}_t = \boldsymbol{\xi}^n)$. Note that the Gaussian basis functions $\phi_n(\boldsymbol{x})$ do not change over time, so the state distribution at time t is represented by the weight vector $\boldsymbol{\alpha}_t$. To propagate any distribution multiple steps through the GP, the basis functions ϕ_n must be calculated only once and the task reduces to sequential updates of the weight vector $\boldsymbol{\alpha}$. As $p(\boldsymbol{x}_t) \approx \boldsymbol{\phi}^{\mathsf{T}} \boldsymbol{\alpha}_t$, the weight vector $\boldsymbol{\alpha}_{t+1}$ is given by

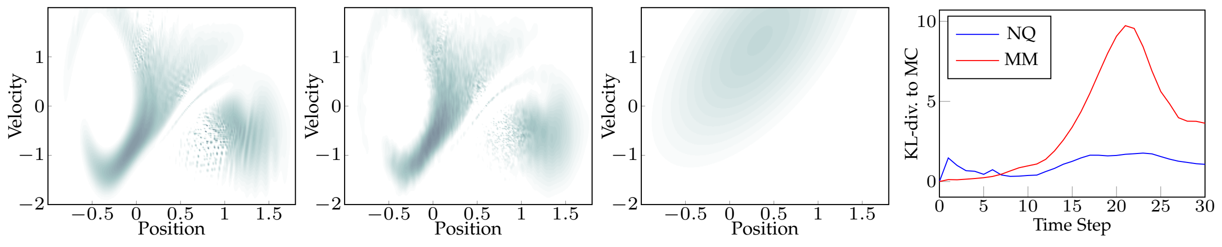
$$\boldsymbol{\alpha}_{t+1} = \operatorname{diag}(\boldsymbol{w}) \Phi \boldsymbol{\alpha}_t = (\operatorname{diag}(\boldsymbol{w}) \Phi)^t \boldsymbol{\alpha}_1 \tag{9}$$

with the matrix Φ, $\Phi_{ij} = \phi_j(\boldsymbol{\xi}^i)$ with $1 \le i, j \le n$, which contains the basis function values at all grid points. In practice, it is helpful to normalize the matrix $\operatorname{diag}(\boldsymbol{w})\Phi$ such that each column sums to 1. In this case, unit vectors will be mapped to unit vectors. This ensures that the approximate state distribution is in fact a probability density, i.e., integrates to 1. Note that the columns of Φ can be computed independently and that, in general, Φ is very sparse. Thus, the computation of Φ and the multiplication in Equation 9 are very well suited for parallel computation, e.g., on a GPU.

To evaluate a policy, we compute

$$R_{\boldsymbol{\pi}}(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}[r(\boldsymbol{x}_t)] \tag{2 revisited}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2879335, IEEE Transactions on Pattern Analysis and Machine Intelligence



(a) Numerical Quadrature (b) Monte Carlo Sampling (c) Moment Matching (d) Approximation Accuracy

Fig. 2. This figure illustrates multi-step-ahead prediction when the input is a distribution ((a)-(c) are taken from [35]). Starting with a normally distributed state centered around the inflection point of the right slope in the mountain car domain (see Sec. 3), the state distribution is approximated at $T = 30$. The plots show the approximate state distribution obtained with numerical quadrature (a), moment matching (c) and the reference Monte Carlo sampling result, computed with 10^5 samples (b). As can be seen, the state distribution significantly differs from a Gaussian. Furthermore, moment matching does not match the first two moments of the state distribution. The distribution obtained with our NQ based approach (a) closely matches the distribution resulting from MC sampling (b). In case of MM (c), the iterative approximation as Gaussian amplifies the initial error. As a result, the distribution obtained by MM (a) is far off the sampled distribution. The KL-divergence between the Monte Carlo result and numerical quadrature (blue)/ MC and moment matching (red) as a function of system time are shown in plot (d).

and due to linearity of the expectation the numerical quadrature approximation results in

$$R_{\boldsymbol{\pi}}(\boldsymbol{\theta}) \approx \sum_{t=0}^{T}\sum_{n=1}^{N} \alpha_{t,n}\mathbb{E}_{\boldsymbol{x}\sim\phi_n}[r(\boldsymbol{x})] =: \widetilde{R}_{\boldsymbol{\pi}}(\boldsymbol{\theta}). \tag{10}$$

Thus, the expected reward in any time step is composed of the rewards for the state to be distributed as one of the Gaussians ϕ_n. When choosing r, hence, one must ensure that expectations with respect to a normally distributed state are analytically tractable, as detailed in Section 2.3.

2.5 Analytic Reward Gradients

When approximating predictions at uncertain inputs with numerical quadrature, the expected long-term reward $\widetilde{R}_{\boldsymbol{\pi}}(\boldsymbol{\theta})$ for following policy $\boldsymbol{\pi_\theta}$ can be computed as given in Equation (10). To improve the policy, we aim to maximize $\widetilde{R}_{\boldsymbol{\pi}}(\boldsymbol{\theta})$ via gradient ascent. In the following, we will derive a closed-form expression for the gradient $\partial\widetilde{R}_{\boldsymbol{\pi}}(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$.

$$\frac{\partial\widetilde{R}_{\boldsymbol{\pi}}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} = \frac{\partial}{\partial\boldsymbol{\theta}}\sum_{t=0}^{T}\sum_{n=1}^{N}\alpha_{t,n}\mathbb{E}_{\boldsymbol{x}\sim\phi_n}[r(\boldsymbol{x})] \tag{11}$$

$$= \sum_{t=0}^{T}\sum_{n=1}^{N}\frac{\partial\alpha_{t,n}}{\partial\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{x}\sim\phi_n}[r(\boldsymbol{x})] \tag{12}$$

$$+ \alpha_{t,n}\frac{\partial}{\partial\boldsymbol{\theta}}\left(\mathbb{E}_{\boldsymbol{x}\sim\phi_n}[r(\boldsymbol{x})]\right) \tag{13}$$

The gradient $\partial/\partial\boldsymbol{\theta}\,\mathbb{E}_{\boldsymbol{x}\sim\phi_n}[r(\boldsymbol{x})]$ was given in [1] for squashed linear and squashed RBF policies. For the first term, we compute $\partial\alpha_{t,n}/\partial\boldsymbol{\theta}$ as follows. Applying Equation (9), we get

$$\frac{\partial\boldsymbol{\alpha}_t}{\partial\boldsymbol{\theta}} = \frac{\partial}{\partial\boldsymbol{\theta}}\left((\mathrm{diag}(\boldsymbol{w})\Phi)^t\boldsymbol{\alpha}_1\right). \tag{14}$$

The derivative of a matrix power A^k with respect to the parameter b can be computed as $\partial A^k(b)/\partial b = \sum_{l=0}^{k-1}A^l(b)\frac{\partial A(b)}{\partial b}A^{k-l-1}(b)$. Note also that $\partial\boldsymbol{\alpha}_1/\partial\boldsymbol{\theta} = 0$ and, thus,

$$\frac{\partial\boldsymbol{\alpha}_t}{\partial\boldsymbol{\theta}} = \frac{\partial}{\partial\boldsymbol{\theta}}\left((\mathrm{diag}(\boldsymbol{w})\Phi)^t\boldsymbol{\alpha}_1\right) = \frac{\partial(\mathrm{diag}(\boldsymbol{w})\Phi)^t}{\partial\boldsymbol{\theta}}\boldsymbol{\alpha}_1 \tag{15}$$

$$= \sum_{l=0}^{t-1}(\mathrm{diag}(\boldsymbol{w})\Phi)^l\frac{\partial(\mathrm{diag}(\boldsymbol{w})\Phi)}{\partial\boldsymbol{\theta}} \tag{16}$$

$$(\mathrm{diag}(\boldsymbol{w})\Phi)^{t-l-1}\boldsymbol{\alpha}_1. \tag{17}$$

Finally, it remains to compute the derivatives of the entries of $\mathrm{diag}(\boldsymbol{w})\Phi$ with respect to $\boldsymbol{\theta}$. As $\boldsymbol{w}$ are the quadrature weights, which do not depend on $\boldsymbol{\theta}$, we compute the derivatives of Φ with respect to $\boldsymbol{\theta}$

$$\frac{\partial\Phi_{ij}}{\partial\boldsymbol{\theta}} = \frac{\partial\phi_j(\boldsymbol{\xi}^i)}{\partial\boldsymbol{\theta}} = \frac{\partial p(\boldsymbol{x}_{t+1} = \boldsymbol{\xi}^j \mid \boldsymbol{x}_t = \boldsymbol{\xi}^i)}{\partial\boldsymbol{\theta}}. \tag{18}$$

Let $\boldsymbol{\mu}_j$ be the mean of ϕ_j and Σ_j its covariance matrix. Note that Σ_j is diagonal, as ϕ_j is the GP predictive distribution at the point $\boldsymbol{\xi}^j$ and, thus, the different state dimensions do not covary.

The derivatives $\partial\boldsymbol{\mu}_j/\partial\boldsymbol{\theta}$ and $\partial\Sigma_j/\partial\boldsymbol{\theta}$ were provided in [1] for squashed linear and squashed RBF policies. Thus, it remains to compute $\partial\phi_j(\boldsymbol{\xi}^i)/\partial\boldsymbol{\mu}_j$ and $\partial\phi_j(\boldsymbol{\xi}^i)/\partial\Sigma_j$ for diagonal Σ_j. Writing $a(\Sigma) = \left((2\pi)^D\det(\Sigma)\right)^{-\frac{1}{2}}$ and $b(\boldsymbol{\mu},\boldsymbol{\xi},\Sigma) = \exp\left((\boldsymbol{\mu}-\boldsymbol{\xi})^\intercal\Sigma^{-1}(\boldsymbol{\mu}-\boldsymbol{\xi})\right)$ these gradients are computed as

$$\frac{\partial\phi_j(\boldsymbol{\xi}^i)}{\partial\boldsymbol{\mu}_j} = \frac{\partial}{\partial\boldsymbol{\mu_j}}\left(a(\Sigma_j)b(\boldsymbol{\mu_j},\boldsymbol{\xi}^i,\Sigma_j)\right)$$
$$= -\phi_j(\boldsymbol{\xi}^i)\Sigma_j^{-1}(\boldsymbol{\mu}_j-\boldsymbol{\xi}^i) \tag{19}$$

$$\frac{\partial\phi_j(\boldsymbol{\xi}^i)}{\partial\Sigma_j} = \frac{\partial}{\partial\Sigma_j}\left(a(\Sigma_j)b(\boldsymbol{\mu_j},\boldsymbol{\xi}^i,\Sigma_j)\right)$$
$$= \frac{\partial a(\Sigma_j)}{\partial\Sigma_j}b(\boldsymbol{\mu_j},\boldsymbol{\xi}^i,\Sigma_j)$$
$$+ a(\Sigma_j)\frac{\partial b(\boldsymbol{\mu_j},\boldsymbol{\xi}^i,\Sigma_j)}{\partial\Sigma_j}$$
$$= -\frac{1}{2}\phi_j(\boldsymbol{\xi}^i)\det(\Sigma_j)^{-1}\Delta$$
$$+ \frac{1}{2}\phi_j(\boldsymbol{\xi}^i)\Sigma_j^{-2}\,\mathrm{diag}(\boldsymbol{\mu}_j-\boldsymbol{\xi}^i)^2 \tag{20}$$

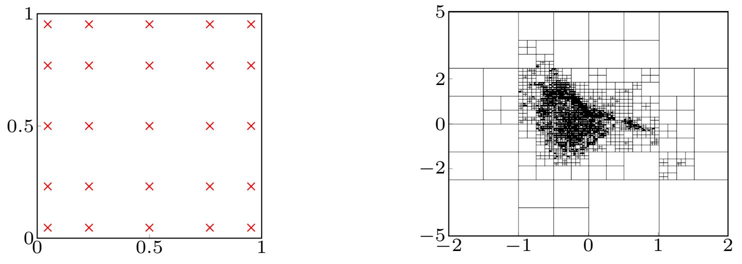


Fig. 3. Construction of suitable quadrature rules. Plot (a) shows the nodes of a Gaussian product quadrature rule on the unit square. Here, the quadrature rule for the unit square was constructed as an outer product of the Gaussian quadrature rule with 5 nodes. A state space partition with approximately 4000 rectangles for the mountain car system obtained with Algorithm 2 is shown in (b). For each rectangle in this partition, a Gaussian product quadrature such as (a) is employed.

with

$$\Delta = \operatorname{diag}\left(\left(\prod_{\substack{m=1,\\ m\neq m'}}^{D} \Sigma_{j,mm}\right)_{m'=1,\dots,D}\right). \tag{21}$$

Combining these gradients with the ones provided in [1] via chain rule, we get $\partial\Phi_{ij}/\partial\boldsymbol{\theta}$. Substituting the result in Equation (17), we compute $\partial\boldsymbol{\alpha}_t/\partial\boldsymbol{\theta}$. Finally, returning to Equation (13) we have computed all components and get a closed-form expression for $\partial\widetilde{R}_{\boldsymbol{\pi}}(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$.

2.6 Construction of Quadrature Rules

Our proposed policy search approach nuQuPS employs numerical quadrature to approximate GP predictions when the input is a distribution. The derived expressions for the approximate distribution and the gradients of the expected long-term reward are valid for any choice of quadrature rule. However, in practice, the choice of a suitable quadrature rule is crucial to the accuracy of the approximate long-term predictions and, thus, to the learning success of nuQuPS. Thus, in the following, we will elaborate on how to construct "good" quadrature rules for a given GP dynamics model. During learning, nuQuPS constructs a quadrature rule in every episode, that is suited to the current GP dynamics model.

For smooth, univariate functions no rule can possibly achieve faster convergence than Gaussian quadrature. Gaussian product quadrature extends univariate Gaussian quadrature to a multivariate rule using a product grid of evaluation points. This construction has some desirable properties, such as positive quadrature weights and readily available quadrature nodes. However, being an outer product of one-dimensional rules, it suffers from the curse of dimensionality and is not optimal anymore. Other types of (partially optimal) quadrature rules have been studied, e.g. [40], however covering the whole space equally well inevitably leads to the curse of dimensionality even for optimal rules. In this paper, we apply numerical quadrature to integrals as in Equation (6) and address the mentioned drawback as follows. Typically, the system trajectories are not uniformly spread over the state space. Instead, they are concentrated in a significantly smaller region. We exploit this observation to improve the efficiency of high dimensional quadrature and cope with the curse of dimensionality. For this purpose, we partition the state space $X = X_1 \sqcup \cdots \sqcup X_L$ as outlined below and apply a multivariate quadrature rule to each obtained subregion X_l. The next state distribution, cf. Equation (6), can be written as

$$\begin{aligned} p(\boldsymbol{x}_{t+1}) &:= \int_X p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t) p(\boldsymbol{x}_t) d\boldsymbol{x}_t \\ &= \sum_{l=1}^{L} \int_{X_l} p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t) p(\boldsymbol{x}_t) d\boldsymbol{x}_t \end{aligned} \tag{22}$$

and, applying a numerical quadrature rule with nodes $\mathbb{X}_l$ for each integral, we get

$$p(\boldsymbol{x}_{t+1}) \approx \sum_{l=1}^{L} \sum_{\boldsymbol{\xi}^n \in \mathbb{X}_l} w_{nl} p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t = \boldsymbol{\xi}^n) p(\boldsymbol{x}_t = \boldsymbol{\xi}^n).$$

Setting $\mathbb{X} = \mathbb{X}_1 \sqcup \cdots \sqcup \mathbb{X}_L$, we recover Equation (7). Thus, composed quadrature rules can be handled just as a single Gaussian product rule. We exploit this fact to construct quadrature rules which are efficient for the computation of next state distributions for our particular GP dynamics model and subregion of policy parameter space. In other words, we aim to find a partition $X = X_1 \sqcup \cdots \sqcup X_L$ of the state space, such that integral (6) is approximated well by the resulting quadrature rule for all policies $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ that will be evaluated during gradient ascent in the current episode. For this purpose, we maintain a partition of the state space, sample trajectories and subdivide regions X_l that were visited and fulfill a certain criterion which we introduce below.

To sample representative system trajectories, we perturb the currently known best policy parameters $\boldsymbol{\theta}^*$ with white noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \Sigma_{\boldsymbol{\epsilon}})$. We sample starting states $\boldsymbol{\tau}_0$ from $p(\boldsymbol{x}_0)$ and compute simulated mean rollouts $\boldsymbol{\tau}_0, \dots, \boldsymbol{\tau}_T$ starting from $\boldsymbol{\tau}_0$ and from the target state $\boldsymbol{x}^d$ with each sampled policy parameter vector $\boldsymbol{\theta}$ and the dynamics GP $\boldsymbol{g}$. Thus, every state in the simulated trajectories will be normally distributed with mean $\boldsymbol{\tau}_i$ and variance $\operatorname{Var}(\boldsymbol{\tau}_i)$. The noise variance $\Sigma_{\boldsymbol{\epsilon}}$ determines how far we expect to differ from the initial policy during gradient ascent. Thus, we start with a high variance and decrease it with the number of episodes.

We subdivide the region X_l, if it does not contain enough quadrature nodes to integrate predictive distributions from

Algorithm 2 Construction of composed quadrature rules

```
Input:
    dynamics GP $\boldsymbol{g}$: $(\boldsymbol{x}_t, \boldsymbol{u}_t) \mapsto \boldsymbol{x}_{t+1}$, start distribution $p(\boldsymbol{x}_0)$, current best policy $\boldsymbol{\pi}_{\boldsymbol{\theta}^*}$, state space $X$, maximum partition size $L_{\max}$, policy noise variance $\Sigma_{\boldsymbol{\epsilon}}$
Output:
    composed quadrature rule with nodes $\mathbb{X}$ and weight vector $\boldsymbol{w}$
1: Initialize partition $X = X_1 \sqcup \cdots \sqcup X_s$ and quadrature $\mathbb{X} = \mathbb{X}_1 \sqcup \cdots \sqcup \mathbb{X}_s$ with $s < L_{\max}$
2: while $s < L_{\max}$ do
3:     Sample starting state $\boldsymbol{\tau}_0 \sim p(\boldsymbol{x}_0)$ and policy parameters $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\theta}^*, \Sigma_{\boldsymbol{\epsilon}})$
4:     Compute rollouts $\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_T$ and $\boldsymbol{x}^d = \boldsymbol{\tau}'_0, \ldots, \boldsymbol{\tau}'_T$ with dynamics GP $\boldsymbol{g}$ and policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$
5:     for $l = 1, \ldots, s$ do
6:         if $\frac{\mathrm{vol}(X_l)}{|\mathbb{X}_l| \min_{\boldsymbol{\tau}_i, \boldsymbol{\tau}'_i \in X_l} \mathrm{Var}(\boldsymbol{\tau}_i)} < 1$ then subdivide $X_l$, add nodes to $\mathbb{X}$ fi
7:     od
8: od
9: return quadrature nodes $\mathbb{X}$ and weights $\boldsymbol{w}$
```

the dynamics GP well. To estimate whether X_l should be divided, we introduce

$$\rho_l(\boldsymbol{\tau}) := \frac{\mathrm{vol}(X_l)}{|\mathbb{X}_l| \min_{\boldsymbol{\tau}_i \in X_l} \mathrm{Var}(\boldsymbol{\tau}_i)} \tag{23}$$

and subdivide X_l if $\rho_l(\boldsymbol{\tau})$ is greater than 1. This criterion relates the higher order derivatives of GP predictions which fall inside X_l with the quadrature node density in X_l.

Constructing a composed quadrature rule with this approach will concentrate most quadrature nodes in state space regions that are visited frequently when following system trajectories. Algorithm 2 summarizes our approach and Figure 3 illustrates the constructed quadrature rules.

3 Empirical Evaluation on Typical Benchmark Problems

We evaluate the proposed algorithm on three benchmark tasks: mountain car, cart-pole swing up and hold and cart-double-pendulum balancing. We compare our results with other model-based approaches in multiple experiments. First, we briefly introduce the two test-beds.

Mountain Car. In the mountain-car domain (see, e.g., [41], [42]), a car starts at some point in a valley landscape and has to reach a certain point on the hill to the right side of the valley and stay there. However, the car's engine is not powerful enough to reach the goal directly from all starting positions. From the points in the valley, the car has to first drive in the opposite direction and gain momentum to reach the goal. The state space has two dimensions: position and velocity of the car, the control signal is limited to $u_{\max} = 4$.

Cart-Pole. In the cart-pole domain [1], a cart with an attached free-swinging pendulum is running on a track of limited length. The goal is to swing the pendulum up and balance it, with the cart in the middle of the track. The state space has four dimensions: position of the cart x, velocity of the cart $\dot{x}$, angle of the pendulum ϑ and the angular velocity $\dot{\vartheta}$. A horizontal force of $u_{\max} = 10$ can be applied to the cart.

Cart-Double-Pendulum. In the inverted double pendulum on a cart balancing task, a double pendulum is mounted on a cart. Starting with an (almost) upright position, the goal is to balance the pendulum and stabilize the system. The state space has six dimensions: position of the cart, velocity of the cart, angles of the two pendulum arms and their angular velocities. A horizontal force with $u_{\max} = 10$ can be applied to the cart.

To evaluate the proposed algorithm (nuQuPS), we perform several experiments on the two test-beds. First, we examine how the approximation of multi-step-ahead predictions affects learning, comparing moment matching to the proposed numerical quadrature on the mountain car task. Second, we perform the cart-pole swing up and balance task. Third, we demonstrate how nuQuPS is capable of learning global policies.

3.1 Learning Control in the Mountain Car Domain

In this experiment, we want to evaluate how the proposed numerical quadrature approximate inference performs in comparison to moment matching and how these approximation methods affect learning. For this purpose, we start with a given, pre-trained GP dynamics model for the mountain car domain. We want to test how well we can learn policies for starting states concentrated in different regions of the state space. Thus, we build a grid in the state space region $[-1, 1] \times [-0.2, 0.2]$ and conduct a policy search for each grid point, choosing $p(\boldsymbol{x}_0)$ as a Gaussian centered at the corresponding grid point. To evaluate the learned policies, we test them on the given GP dynamics via Monte Carlo. From each starting distribution, we compute 100 sample rollouts employing the learned policy. We average the distance of the final rollout point to the target state over these rollouts. We color each cell according to the computed average distance, which gives a map of the learning success. Here, lower values (and darker colors) indicate better performance of the policy.

We perform this experiment computing the expected long-term reward and its gradient with respect to the policy parameters with (i) numerical quadrature and (ii) moment matching. For the start distributions, we choose the covariance matrix $0.0025I$. For every starting distribution, we set the time horizon $T = 30$ and learn a squashed linear policy $\boldsymbol{\pi}(\boldsymbol{x}) = u_{\max}\sigma(M\boldsymbol{x} + b)$, cf. Section 2.3. For the numerical quadrature, we employed our quadrature rule construction method (Algorithm 2) with an outer product of the Gaussian rule with 5 nodes in each subregion.

Figure 4, plots (a) and (b), show the results. As can be seen, numerical quadrature provides a more accurate estimate of the expected reward and its gradient and, thus, is more reliable in learning a successful policy. However, maximizing the expected long-term reward is a non-convex optimization task and, thus, optimization may not find a successful policy in all cases.

Note, that solving the mountain car task with a squashed linear policy requires to maintain a delicate balance between gaining momentum for the car to drive up the hill and slowing the car down at the right time to hold it at the target. To evaluate whether we succeeded to learn policies

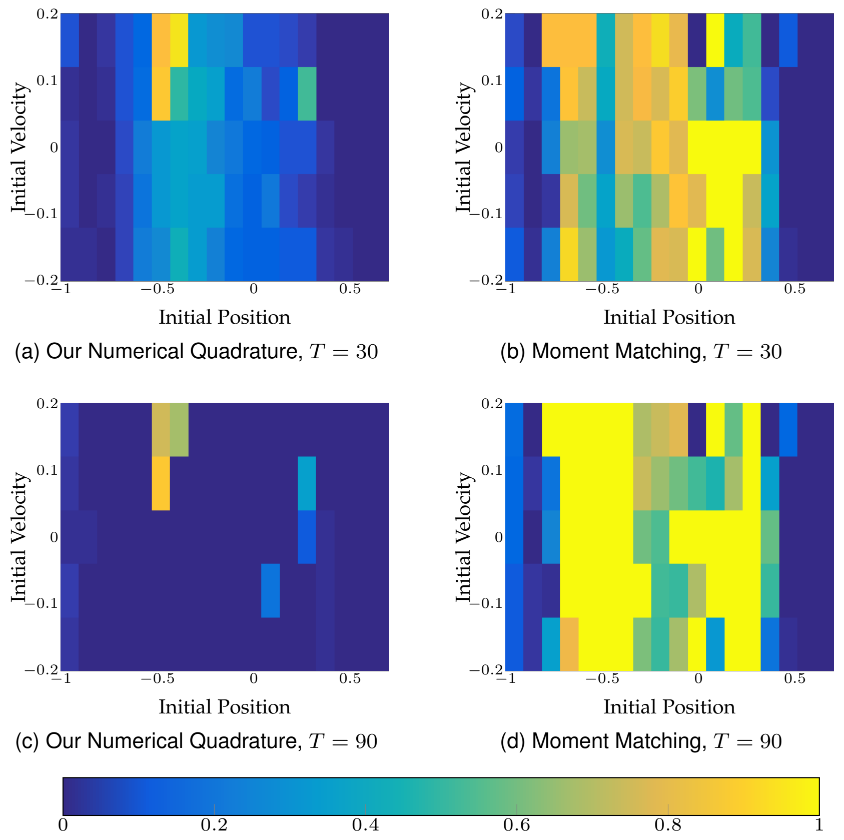


Fig. 4. Comparison of the numerical quadrature and moment matching approximations for policy search. Given a pre-trained GP dynamics model for the mountain car domain, a squashed linear policy was learned starting in each cell. Policy values and gradients were computed with numerical quadrature (plots (a) and (c)) and moment matching (plots (b) and (d)). With the learned policy, rollouts from the corresponding cell were computed with time horizons $T = 30$ and $T = 90$. The cells are colored according to the average distance of the rollout final state to the target state (darker is better).

that are capable of both accelerating the car sufficiently and hold it, as opposed to the local optimum of accelerating the car as much as possible and overshoot, we also test the learned policies with a larger time horizon $T = 90$. The results are shown in Figure 4, plots (c) and (d). As can be seen, in most cases numerical quadrature finds a policy that can both accelerate and hold the car. In contrast, moment matching typically fails at this task.

3.2 Learning Control in the Cart-Pole and Cart-Double-Pendulum Domains

To evaluate overall performance of the proposed nuQuPS algorithm, we learn to swing up and balance the pendulum in the cart-pole (CP) domain. Starting with the cart standing in the middle of the tracks, pendulum down, we perform five experiments with different initial policies. The starting state variance was set to 10^{-6} and the time horizon to $T = 40$. As a quadrature rule, the CN:3-1 rule [40], [43] was employed. We learn squashed linear policies for this task. Note that a linear policy cannot swing up and then balance the pendulum on the cart. However, there are squashed linear policies that are capable of this task.

Secondly, we learn to balance the double pendulum on a cart (CDP) with nuQuPS and PILCO. For this task, we also learn a squashed linear policy to stabilize the system. The

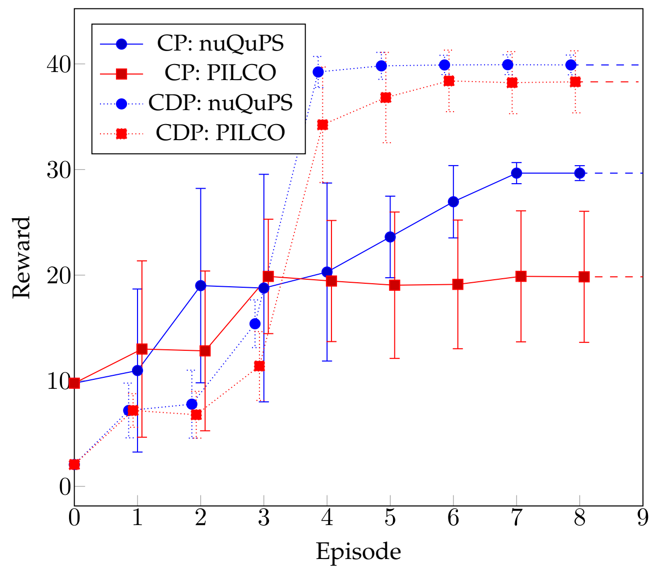


Fig. 5. Average reward with standard deviation as a function of system interaction time for the cart-pole swing up and hold and the cart-double-pole balancing tasks. One episode equals 4 seconds of system interaction time. For both tasks, squashed linear policies were learned with our nuQuPS approach (blue) and PILCO (red). All results were averaged over five trials with different initial policies.

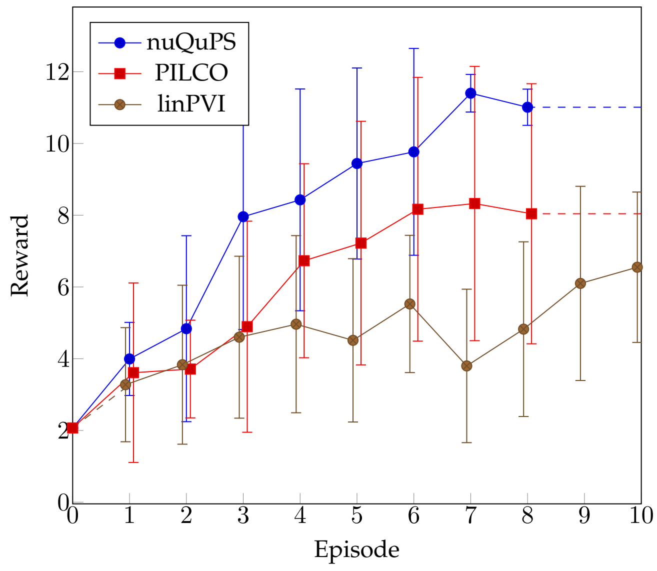


Fig. 6. Average reward with standard deviation as a function of system interaction time for the mountain car task. One Episode equals 3 seconds of system interaction time. The task was to learn a squashed linear policy for all car starting positions in $[-1, 1] \times [-0.1, 0.1]$. For nuQuPS, a uniform distribution over this region was chosen as the starting state. For PILCO, we chose 10 starting points on a regular grid in the starting region. PVI obtained rollouts with the starting point sampled from the uniform starting distribution at the end of each episode. After learning, a squashed, linear policy was fitted with a least squares approach.

starting state variance was set to 10^{-4} and the time horizon to $T = 40$. For the quadrature, we used Algorithm 2 and the CN:3-1 rule [40], [43].

Figure 5 shows the average reward per episode obtained with nuQuPS in comparison to the PILCO algorithm for the cart-pole (solid lines) and cart-double-pendulum (dotted lines). The policies from each episode were evaluated on 100 rollouts, where the starting point was drawn from the start distribution $p(\boldsymbol{x}_0)$.

3.3 Learning Global Policies for the Mountain Car Task

In this experiment, our goal is to learn a policy that can successfully solve the mountain car task for a continuous region of starting points. One major advantage of nuQuPS is that it can handle arbitrary starting state distributions. For our task, we choose the starting state to be uniformly distributed on $[-1, -0.1] \times [1, 0.1]$.

We compare the learning success with two other model-based approaches: probabilistic value iteration (PVI) [29] and PILCO [1]. PVI learns a global value function and, thus, can handle our uniform starting state distribution. For the rollout performed at the end of an episode, the starting point is sampled from $p(\boldsymbol{x}_0)$. As a value iteration approach, PVI does not assume a certain parametric form of the policy, which enables highly flexible and complex policies. However, to compute the learned optimal actions, an argmax must be performed for every time step, which is computationally challenging and often impractical. To overcome this drawback and also make PVI comparable to the other algorithms, after learning, we fit a squashed linear policy with least squares. The policy obtained with this approach will be named linPVI to distinguish from the optimal policy, that can be computed from the value function obtained with PVI.

To learn a policy for all starting states in $[-1, -0.1] \times [1, 0.1]$ with PILCO, we follow the approach in [20]. We distribute ten starting points over $[-1, -0.1] \times [1, 0.1]$ and maximize the mean expected long-term reward of these points with PILCO.

The results are shown in Figure 6. For each policy, the obtained reward was averaged over 3000 rollouts with starting points sampled from the initial state distribution. We perform five trials with different initial policies for the three approaches. As can be seen, nuQuPS outperforms PILCO and PVI on this task. We found that PILCO is prone to local optima, which leads to a high variance in learning success. PVI, on the other hand, shows slower convergence, as learning a global value function is a significantly harder task than finding policy parameters for a given starting state distribution. In addition, in PVI the support points must be chosen, where the value function will be computed and used as training data for the value function model. This selection of support points is a nontrivial task and can highly affect the learning success.

4 Conclusion

Learning control is promising as it allows to drastically reduce the amount of expert knowledge, that is required otherwise. Model-based approaches are particularly appealing since they are highly data efficient, especially when probabilistic models are employed to account for the inherently uncertain dynamics estimates. However, long-term predictions with such models typically become intractable, introducing the need for approximations. In this paper, we show that the accuracy of the chosen approximation method significantly affects learning and propose numerical quadrature to approximate long-term predictions. We conclude the paper with a short summary of the main contributions and a brief outlook on possible future work in this direction.

4.1 Summary of Contributions

In this paper, we introduced nuQuPS, a model-based policy search approach, that makes use of Gaussian processes as dynamics models. To propagate uncertainties through the GP dynamics model, nuQuPS employs numerical quadrature. Numerical quadrature for approximate inference can model complex distributions, e.g., with multiple modes. The numerical quadrature approximation can be parallelized straightforwardly and, thus, be computed efficiently. Furthermore, we provided analytic gradients that can be used for policy search. With these analytic gradients, policy improvement can be performed with any gradient based optimization scheme.

Due to its flexibility, numerical quadrature provides highly accurate approximations even for complex distributions, e.g., when the input distribution has high variance. This high accuracy significantly speeds up learning and results in enhanced robustness and data efficiency. Additionally, numerical quadrature can handle arbitrary, non-Gaussian starting state distributions, e.g., a uniform distribution over all possible starting states. As a result, nuQuPS provides a principled way to learn policies, that are suitable, e.g., for all feasible starting states. Thus, nuQuPS combines the ability to learn global policies and remains scalable to high-dimensional problems.

Combining Bayesian averaging with high-performance uncertainty propagation, nuQuPS achieves remarkable data-efficiency on all tested tasks. It is highly robust to different choices of initial policy. Evaluation on three simulated test-beds demonstrates nuQuPSs superior performance.

4.2 Discussion and Next Steps

The proposed nuQuPS algorithm allows to robustly learn global policies with unprecedented data-efficiency. However, there remain several open questions. First of all, while we evaluated nuQuPS on multiple benchmark domains in simulation, it has not yet been applied on a real system and will be adressed in future work.

Second, to learn a policy, nuQuPS proceeds in an episodic setting, acquiring more data from system interactions, updating its dynamics model and re-optimizing the policy to suit the updated dynamics model. In all tests we performed, this procedure converged after a few episodes. However, as for most policy search approaches on continuous state and action spaces, there is no guarantee that nuQuPS will converge. In nuQuPS, maximizing the expected long-term reward is a non-convex optimization task and, thus, no guarantees can be given. However, even when assuming that the global optimum is found in every episode, convergence of nuQuPS remains an open problem.

Third, our nuQuPS approach, as most previous policy search approaches, optimizes the policy for the case that there are no external disturbances. Robust control, however, considers the problem of learning a policy that is successful even when external disturbances are present. To solve this task, assumptions e.g., on the parametric form of the disturbance are made and stability analysis is employed. Recently, stability analysis for learned GP dynamics and policies was provided [15]. This stability analysis combined with a high performance policy search approach as the proposed nuQuPS could open the door for learning robust control.

Finally, numerical quadrature has proven to provide highly accurate approximate multi-step-ahead predictions, that greatly speed up policy learning. These high-quality approximations could as well support other Bayesian model-based learning approaches. Especially for value iteration based on GP dynamics models [26], [28], [29], numerical quadrature could be promising to cope with the intractability of the model.

References

[1] M. Deisenroth, D. Fox, and C. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 2, pp. 408–423, 2015.

[2] Y. Pan and E. Theodorou, "Probabilistic differential dynamic programming," in *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1907–1915.

[3] E. Klenske, M. Zeilinger, B. Schölkopf, and P. Hennig, "Nonparametric dynamics estimation for time periodic systems," in *51st Annual Allerton Conference on Communication, Control, and Computing*. IEEE, 2013, pp. 486–493.

[4] J. Maciejowski and X. Yang, "Fault tolerant control using gaussian processes and model predictive control," in *Conference on Control and Fault-Tolerant Systems (SysTol 2013)*. IEEE, 2013, pp. 1–12.

[5] D. Nguyen-Tuong and J. Peters, "Model learning in robotics: a survey," *Cognitive Processing*, no. 4, 2011.

[6] Y. Engel, P. Szabo, and D. Volkinshtein, "Learning to control an octopus arm with gaussian process temporal difference methods," in *Advances in Neural Information Processing Systems 18 (NIPS 2005)*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. MIT Press, 2006, pp. 347–354.

[7] J. Kocijan, R. Murray-Smith, C. Rasmussen, and A. Girard, "Gaussian process model based predictive control," in *Proceedings of the American Control Conference, (ACC 2004).*, vol. 3. IEEE, 2004, pp. 2214–2219.

[8] B. Bischoff, D. Nguyen-Tuong, T. Koller, H. Markert, and A. Knoll, "Learning throttle valve control using policy search," in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*, 2013.

[9] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Proceedings of 1997 IEEE International Conference on Robotics and Automation*. IEEE, 1997, pp. 3557–3564.

[10] L. Kuvayev, "Model-based reinforcement learning with an approximate, learned model," in *in Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, 1997.

[11] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, no. 1, pp. 103–130, 1993.

[12] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *In Proceedings of the Seventh International Conference on Machine Learning (ICML 1990)*. Morgan Kaufmann, 1990, pp. 216–224.

[13] J. Schneider, "Exploiting model uncertainty estimates for safe dynamic control learning," in *Advances in Neural Information Processing Systems 9 (NIPS 1996)*.

[14] T. Beckers and S. Hirche, "Stability of gaussian process state space models," in *Proceedings of the European Control Conference*. IEEE, 2016, pp. 2275–2281.

[15] J. Vinogradska, B. Bischoff, D. Nguyen-Tuong, A. Romer, H. Schmidt, and J. Peters, "Stability of controllers for gaussian process forward models," in *Proceedings of the 33nd International Conference on Machine Learning (ICML 2016)*, 2016, pp. 545–554.

[16] A. Girard, C. E. Rasmussen, J. Q. Candela, and R. Murray-Smith, "Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting," in *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, 2002, pp. 529–536.

[17] J. Ko and D. Fox, "GP-BayesFilters: Bayesian filtering using gaussian process prediction and observation models," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008.

[18] J. Quiñonero-Candela, A. Girard, J. Larsen, and C. Rasmussen, "Propagation of uncertainty in bayesian kernel models - application to multiple-step ahead forecasting," in *International Conference on Acoustics, Speech and Signal Processing*, 2003, pp. 701–704, vol. 2.

[19] M. Ghavamzadeh and Y. Engel, "Bayesian policy gradient algorithms," in *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 457–464.

[20] M. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *Proceedings of 2014 IEEE International Conference on Robotics and Automation*. IEEE, 2014, pp. 3876–3881.

[21] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, 2005.

[22] K. Zhou and J. Doyle, *Essentials of Robust Control*, ser. Prentice Hall Modular Series for Eng. Prentice Hall, 1998.

[23] K. Narendra and A. Annaswamy, *Stable Adaptive Systems*, ser. Dover Books on Electrical Engineering. Dover Publications, 2012.

[24] G. Tao, *Adaptive Control Design and Analysis*. John Wiley & Sons, Inc., 2003.

[25] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-efficient generalization of robot skills with contextual policy search." in *AAAI*, 2013.

[26] C. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems 16 (NIPS 2003)*. MIT Press, 2004, pp. 751–759.

[27] A. Rottmann and W. Burgard, "Adaptive autonomous control using online value iteration with gaussian processes," in *Proceedings*

of the 2009 IEEE International Conference on Robotics and Automation. IEEE, 2009, pp. 2106–2111.

[28] M. Deisenroth, C. Rasmussen, and J. Peters, "Gaussian process dynamic programming," *Neurocomputing*, vol. 72, no. 7-9, pp. 1508–1524, Mar. 2009.

[29] B. Bischoff, D. Nguyen-Tuong, H. Markert, and A. Knoll, "Learning control under uncertainty: A probabilistic value-iteration approach," in *21st European Symposium on Artificial Neural Networks, ESANN 2013*, 2013.

[30] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-box data-efficient policy search for robotics," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 51–58.

[31] A. Doerr, C. Daniel, D. Nguyen-Tuong, A. Marco, S. Schaal, M. Toussaint, and S. Trimpe, "Optimizing long-term predictions for model-based policy search," in *Proceedings of the 1st Annual Conference on Robot Learning (CoRL 2017)*, 2017, pp. 227–238.

[32] Y. Chen, M. Welling, and A. Smola, "Super-samples from kernel herding," in *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, ser. UAI'10. Arlington, Virginia, United States: AUAI Press, 2010, pp. 109–116.

[33] F. Bach, S. Lacoste-Julien, and G. Obozinski, "On the equivalence between herding and conditional gradient algorithms," in *Proceedings of the 29th International Coference on International Conference on Machine Learning (ICML 2012)*. USA: Omnipress, 2012, pp. 1355–1362.

[34] F. Huszár and D. Duvenaud, "Optimally-weighted herding is bayesian quadrature," in *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI 2012)*. Arlington, Virginia, United States: AUAI Press, 2012, pp. 377–386.

[35] J. Vinogradska, B. Bischoff, D. Nguyen-Tuong, and J. Peters, "Stability of controllers for gaussian process dynamics," *Journal of Machine Learning Research*, vol. 18, no. 100, pp. 1–37, 2017.

[36] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[37] I. Gradshteyn, A. Jeffrey, and I. Ryzhik, *Table of Integrals, Series, and Products*. Academic Press, 1996.

[38] E. Süli and D. Mayers, *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[39] B. Skrainka and K. Judd, "High performance quadrature rules: How numerical integration affects a popular model of product differentiation," *Available at SSRN 1870703*, 2011.

[40] A. Stroud, *Approximate calculation of multiple integrals*, ser. Prentice-Hall series in automatic computation. Prentice-Hall, 1971.

[41] A. W. Moore and C. G. Atkeson, "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces," *Machine Learning*, vol. 21, no. 3, pp. 199–233, 1995.

[42] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

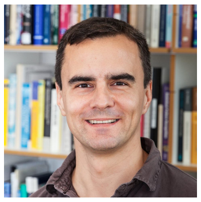
[43] J. Burkardt, "Stroud – numerical integration in m dimensions," https://people.sc.fsu.edu/~jburkardt/m_src/stroud/stroud.html, 2014.

Julia Vinogradska received her B.Sc. and M.Sc. Mathematics degrees from the Universität Stuttgart. She has received the Stiftung Werner von Siemens Ring Jungwissenschaftler (junior scientist) award for the research conducted during her PhD studies at the Intelligent Autnonomous Systems Group, TU Darmstadt and Bosch Center for Artificial Intelligence. Currently, she is a research scientist at the Bosch Center for AI in Renningen, Germany. Julia's research focuses on reinforcement learning in continuous domains and decision making under uncertainty.

Bastian Bischoff is heading a research group on deep learning for perception at the Bosch Center for Artificial Intelligence in Renningen, Germany. He has received his Diplom (comparable to M.Sc.) in computer science from the Universität Stuttgart, Stuttgart, Germany, and his PhD from TU Munich, Munich, Germany, in collaboration with Bosch on "Reinforcement Learning for Industrial Applications". At Bosch Center for Artificial Intelligence, Bastian and his team work towards explainable perception from various sensor modalities to enable autonomous driving.

Jan Achterhold received the B.Sc. (2015) in electrical engineering, information technology and computer engineering from RWTH Aachen University, Germany, where he is now pursuing the M.Sc. in systems engineering and automation. From October 2016 to March 2017 he interned at Bosch Corporate Research, Renningen, Germany in the field of reinforcement learning. He is currently working on his master's thesis at the Bosch Center for Artificial Intelligence, Renningen, Germany. His research interests include machine learning, control theory and the intersection of both.

Torsten Koller Koller was born in Stuttgart, Germany on April 21, 1991. He received the B.Sc. in Mathematics from Technical University of Munich, Munich, Germany. During his studies, Torsten worked with the Robert Bosch GmbH Corporate Research Department in several research projects. He worked as a student assistant at the Brain-State Decoding Lab, University of Freiburg, Freiburg, Germany from April 2015 to August 2016. Torsten Koller is a graduate student at University of Freiburg pursuing the M.Sc. degree in Computer Science. He is currently writing his master's thesis at the Learning & Adaptive Systems Group at ETH Zürich, Switzerland.



Jan Peters is a full professor (W3) for Intelligent Autonomous Systems at the Computer Science Department of the Technische Universität Darmstadt and at the same time a senior research scientist and group leader at the Max-Planck Institute for Intelligent Systems, where he heads the interdepartmental Robot Learning Group. Jan Peters has received the Dick Volz Best 2007 US PhD Thesis Runner-Up Award, the Robotics: Science & Systems - Early Career Spotlight, the INNS Young Investigator Award, and the IEEE Robotics & Automation Society's Early Career Award. Recently, he received an ERC Starting Grant.

Jan Peters has studied Computer Science, Electrical, Mechanical and Control Engineering at TU Munich and FernUni Hagen in Germany, at the National University of Singapore (NUS) and the University of Southern California (USC). He has received four Master's degrees in these disciplines as well as a Computer Science PhD from USC. Jan Peters has performed research in Germany at DLR, TU Munich and the Max Planck Institute for Biological Cybernetics (in addition to the institutions above), in Japan at the Advanced Telecommunication Research Center (ATR), at USC and at both NUS and Siemens Advanced Engineering in Singapore.