Approximate Dynamic Programming with Gaussian Processes

Marc P. Deisenroth[1,2], Jan Peters[2], and Carl E. Rasmussen[1,2]

***Abstract*— In general, it is difficult to determine an optimal closed-loop policy in nonlinear control problems with continuous-valued state and control domains. Hence, approximations are often inevitable. The standard method of discretizing states and controls suffers from the curse of dimensionality and strongly depends on the chosen temporal sampling rate. In this paper, we introduce Gaussian process dynamic programming (GPDP) and determine an approximate globally optimal closed-loop policy. In GPDP, value functions in the Bellman recursion of the dynamic programming algorithm are modeled using Gaussian processes. GPDP returns an optimal state-feedback for a finite set of states. Based on these outcomes, we learn a possibly discontinuous closed-loop policy on the entire state space by switching between two independently trained Gaussian processes. A binary classifier selects one Gaussian process to predict the optimal control signal. We show that GPDP is able to yield an almost optimal solution to an LQ problem using few sample points. Moreover, we successfully apply GPDP to the underpowered pendulum swing up, a complex nonlinear control problem.**

I. Introduction

Optimal control is one of the most intuitive setups for specifying control policies: one simply specifies a cost function to be minimized. Due to the work of Bellman, Howard, Kalman, and others, dynamic programming (DP) became the standard approach to solve optimal control problems. However, only in the case of linear systems with quadratic loss and Gaussian noise are exact solutions known [1]. For nonlinear systems the solution to the optimal control problem is more difficult and optimal closed-loop policies cannot be obtained in general. Thus, approximations have to be used to find suitable, suboptimal solutions.

One standard method to approximate a closed-loop policy (not just an open-loop optimal trajectory) for the nonlinear optimal control problem is based on discretization of state and control spaces, which reduce continuous-valued problems to discrete ones. Unfortunately, the resulting discrete algorithm suffers from the curse of dimensionality. Furthermore, the dynamic behavior of the system is strongly affected by the relation between time-, state-, and control discretization. In contrast, DP-based methods with function approximation aim to solve the problem directly in continuous domains. Function approximators generalize value functions to continuous-valued state spaces, while usually keeping the action domain discrete. In [2], parametric value function approximators are suggested to bypass the curse of dimensionality. Parametric models, however, can embody unjustified assumptions meaning that even in the limit of infinite data there is a risk of modeling the underlying function incorrectly. Kernel-based function approximations for reinforcement learning (RL) are introduced and proven to be consistent under mild assumptions in [3], [4]. Gaussian processes (GPs) are a kernel machines and provide a state-of-the-art nonparametric Bayesian regression framework commonly used in machine learning [5].

To date, GPs have been used in control to derive alternative solutions to the optimal control problem. In [6] a nonlinear discrete-time system is modeled using GPs. According to [7] this model is used for predictions multiple time steps into future. In [8], an optimal controller following reference trajectories is derived. A closed-form evaluation of the value function of a nonlinear, discrete-time system with continuous state and control spaces is presented. The system dynamics are modeled using a GP, which allows for analytic policy evaluation. Updating the policy according to gradient information completes policy iteration. One Bayesian approach to model-free policy iteration is proposed in [9]. The authors suggest using GPs to solve the policy evaluation and policy improvement steps, respectively. Rewards and transitions are considered as stochastic.

Although the use of GPs in policy iteration was suggested for instance in [8], [9], their combination with value iteration methods in fully observable Markov decision processes (MDPs) has not been explored in the literature to the best of our knowledge. In this paper, we introduce Gaussian process dynamic programming (GPDP). GPDP is an approximate dynamic programming method, where value functions in the DP recursion are modeled by GPs. Thus, we are able consider continuous-valued states and controls and bypass discretization problems. GPDP yields an approximately optimal state-feedback for a finite set of states. These state-feedback values are generalized to a closed-loop policy defined on the entire continuous-valued state space. To model possibly discontinuous policies properly, we independently train two GPs. For any new query point, a binary classification problem has to be solved to select the GP that predicts the corresponding optimal control signal.

The remainder of the paper is organized as follows. In Section II, parallels between optimal control and reinforcement learning are pointed out to motivate machine learning techniques in control. The section is concluded by an introduction into Gaussian processes. In Section III, we introduce GPDP and describe how to determine a possibly discontinuous closed-loop policy on the entire state space. In Section IV, we apply GPDP to a linear quadratic (LQ) problem so that it yields the optimal solution given sufficient quantities of data. Moreover, we successfully apply GPDP to the underpowered pendulum swing-up problem. In the

[1] Department of Engineering, University of Cambridge, Cambridge, UK
[2] Max Planck Institute for Biological Cybernetics, Tübingen, Germany

context of this problem, we briefly discuss the computational and memory requirements of GPDP and standard DP. Finally, in Section V, results of the paper are summarized and a survey of future work is given.

II. Background

A. Optimal Control and Reinforcement Learning

Both optimal control and reinforcement learning aim at finding a policy that optimizes a performance measure. A policy $\pi : \mathcal{X} \to \mathcal{U}$ is a mapping from state space $\mathcal{X} \subseteq \mathbb{R}^{n_x}$ into control space $\mathcal{U} \subseteq \mathbb{R}^{n_u}$ that assigns a control action to each state. In many cases, the performance measure is defined as the expected loss over a certain time interval. For a state $\mathbf{x}_0 \in \mathcal{X}$ and a policy π, the (discounted) expected cumulative loss of a finite N-step optimization horizon is

$$V_0^\pi(\mathbf{x}_0) \coloneqq \mathrm{E}\left[\gamma^N g_{\text{term}}(\mathbf{x}_N) + \sum_{k=0}^{N-1} \gamma^k g(\mathbf{x}_k, \mathbf{u}_k)\right], \quad (1)$$

where k indexes discrete time. Here, $\mathbf{u} \coloneqq \pi(\mathbf{x})$ is the control chosen under policy π. The function g_{term} is a control-independent terminal loss incurred at time step N. The immediate loss is denoted by $g(\mathbf{x}_k, \mathbf{u}_k)$. The discount factor $\gamma \in [0, 1]$ weights future losses. An optimal policy π_0^* for the N-step problem minimizes (1) for any initial state $\mathbf{x}_0$. The associated bounded state-value function V satisfies Bellman's equation

$$V(\mathbf{x}) = \min_{\mathbf{u} \in \mathcal{U}} \left(g(\mathbf{x}, \mathbf{u}) + \gamma \mathop{\mathrm{E}}_{\mathbf{x}'}[\, V(\mathbf{x}') | \mathbf{x}, \mathbf{u}]\right) \quad (2)$$

for all $\mathbf{x} \in \mathcal{X}$. Here, the successor state for a given state-action pair $(\mathbf{x}, \mathbf{u})$ is denoted by $\mathbf{x}'$. The state-action value function Q is defined by

$$Q(\mathbf{x}, \mathbf{u}) = g(\mathbf{x}, \mathbf{u}) + \gamma \mathop{\mathrm{E}}_{\mathbf{x}'}[\, V(\mathbf{x}') | \mathbf{x}, \mathbf{u}]\,, \quad (3)$$

such that $V(\mathbf{x}) = \min_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u})$ for all $\mathbf{x}$. In general, finding an optimal policy π^* is hard. Assuming time-additive losses and an underlying MDP, the minimal expected cumulative loss can be calculated by dynamic programming. DP determines the optimal state-value function V by using the Bellman recursion

$$V_k(\mathbf{x}_i) = \min_{\mathbf{u}_j \in \mathcal{U}} \left(g(\mathbf{x}_i, \mathbf{u}_j) + \gamma\, \mathrm{E}\left[V_{k+1}(\mathbf{x}_{k+1}) | \mathbf{x}_i, \mathbf{u}_j\right]\right) \quad (4)$$

as a fixed-point iteration scheme, where $\mathbf{x}_i \in \mathcal{X}, \mathbf{u}_j \in \mathcal{U}$, and $k = N-1, \ldots, 0$. The recursion is initialized by setting $V_N \coloneqq g_{\text{term}}$. The state-value function $V_k(\mathbf{x}_i)$ is the minimal expected loss over an $N-k$ step optimization horizon starting from state $\mathbf{x}_i$. Analogously to (4), a recursive approximation of Q by Q_k can be defined.

In contrast to optimal control theory, the standard setup of reinforcement learning is more general. In RL, we usually do not assume known transition dynamics and losses. General RL algorithms have to treat these quantities as random variables. However, if RL algorithms are applied to a fully known MDP, it can be considered as equivalent to optimal control. The Bellman recursion and, therefore, all related algorithms can be used to solve this problem.

For further details on optimal control, dynamic programming, and reinforcement learning, we refer to [1], [2], [10].

In this paper, we consider a more general setting in which an expert rates state-dependent controls with corresponding costs. Perception of these costs can be corrupted by noise. This setting can be translated into a noisy immediate loss signal that is not determined by the controller itself, but externally given by an expert.

Since DP is often inapplicable to nonlinear dynamics in continuous-valued state and control domains, suitable approximations are necessary to find a good policy. The similarity of optimal control and reinforcement learning allows for the combination of approximation techniques from both fields to provide richer solutions.

B. Gaussian Processes for Regression

In machine learning, Gaussian processes are used to infer latent functions from a set of observed function values and prior assumptions. One way to think of a GP is as a distribution over functions. Then, inference takes place directly in function space [5]. A GP is completely specified by a mean function $m(\,\cdot\,)$ and a positive semidefinite covariance function $k(\,\cdot\,,\,\cdot\,)$, also called a kernel. We denote a latent function f that is modeled by a GP as $f \sim \mathcal{GP}(m, k)$.

A GP model is not restricted to a certain parametric class of functions, such as polynomials. Instead, all function classes that share the same prior assumptions are covered. Most of the prior assumptions are implicitly encoded in the choice of the covariance function. A common choice is the squared exponential (SE) covariance function

$$k_{\text{SE}}(\mathbf{x}, \mathbf{x}') \coloneqq s^2 \exp\left(-0.5\,(\mathbf{x} - \mathbf{x}')^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{x}')\right), \quad (5)$$

which reflects the prior belief that we expect the latent function to be smooth [5]. This means, the closer the two inputs $\mathbf{x}$ and $\mathbf{x}'$ are the more correlated the corresponding function values $f(\mathbf{x}), f(\mathbf{x}')$ will be. The degree of correlation is determined through the length-scale parameters ℓ_i in $\boldsymbol{\Sigma} = \mathrm{diag}([\ell_1^{-2}, \ldots, \ell_{n_x}^{-2}])$. In (5), the variance of the underlying function is denoted by s^2.

In case of noisy measurements, a function value is assumed to be given by $y = f(\mathbf{x}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is independent, zero-mean Gaussian noise with variance σ_ε^2. The parameters of the covariance function and the unknown noise variance are concatenated in a hyperparameter vector $\boldsymbol{\theta} \coloneqq [\ell_1, \ldots, \ell_{n_x}, s, \sigma_\varepsilon]^T$. Based on training data, this vector is optimized and yields the least complex model that explains the data [5]. Conditioned on the training data, the predictive distribution of the function value $f_* = f(\mathbf{x}_*)$ for a new query point $\mathbf{x}_*$ is Gaussian with mean and variance given by

$$\mu(f_*) = k(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1} \mathbf{y}\,, \quad (6)$$

$$\sigma^2(f_*) = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1} k(\mathbf{X}, \mathbf{x}_*)\,. \quad (7)$$

The training data is given by the matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ of training inputs and the vector $\mathbf{y} = [y_1, \ldots, y_n]^T$ of corresponding training outputs (observations). $\mathbf{K}$ is the kernel matrix with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The first term in (7) is the

Algorithm 1 GPDP

1: **input:** $f, \mathcal{X}, \mathcal{U}$
2: $V_N(\mathcal{X}) = g_{\text{term}}(\mathcal{X})$ ▷ terminal loss
3: $V_N(\,\cdot\,) \sim \mathcal{GP}_v(m_v, k_v)$ ▷ GP model for V_N
4: **for** $k = N-1$ to 0 **do** ▷ recursively
5: **for** all $\mathbf{x}_i \in \mathcal{X}$ **do** ▷ for all states
6: $Q_k(\mathbf{x}_i, \mathcal{U}) = g(\mathbf{x}_i, \mathcal{U}) + \gamma\, \mathrm{E}[V_{k+1}(\mathbf{x}_{k+1}) | \mathbf{x}_i, \mathcal{U}, f] + \varepsilon$
7: $Q_k(\mathbf{x}_i, \,\cdot\,) \sim \mathcal{GP}_q(m_q, k_q)$ ▷ GP model for Q_k
8: $\pi_k^*(\mathbf{x}_i) \in \arg\min_{\mathbf{u}} Q_k(\mathbf{x}_i, \mathbf{u})$
9: $V_k(\mathbf{x}_i) = Q_k\big(\mathbf{x}_i, \pi_k^*(\mathbf{x}_i)\big)$
10: **end for**
11: $V_k(\,\cdot\,) \sim \mathcal{GP}_v(m_v, k_v)$ ▷ GP model for V_k
12: **end for**
13: **return** $\pi^*(\mathcal{X})$ ▷ return optimal state-feedback for $\mathcal{X}$

prior variance of $f(\mathbf{x}_*)$. The second term reduces the prior variance by a non-negative value that expresses how much information is transferred from the training set to $f(\mathbf{x}_*)$.

GP regression using the SE covariance function is equivalent to Bayesian linear regression with infinitely many Gaussian basis functions. Thus, GPs are a practical realization of a universal function approximator. In contrast to common regression methods, Bayesian inference with GPs yields confidence information through the predictive variance (7).

III. Optimal Control with Gaussian Processes

In the following, we consider a nonlinear, deterministic, discrete-time system $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$ and noisy measurements of the immediate loss signal

$$g(\mathbf{x}_k, \mathbf{u}_k) + \varepsilon\,, \tag{8}$$

where ε is independent, zero-mean Gaussian noise with unknown variance σ_ε^2. In Section III-A, we introduce Gaussian process dynamic programming. In Section III-B, we generalize the optimal state-feedback returned by GPDP to a possibly discontinuous policy on the entire state space.

A. Gaussian Process Dynamic Programming

The key idea of GPDP is to model both latent value functions V_k and Q_k in the DP recursion by Gaussian processes. The corresponding GP models are

$$V_k(\,\cdot\,) \sim \mathcal{GP}_v(m_v, k_v)\,,$$
$$Q_k(\mathbf{x}, \,\cdot\,) \sim \mathcal{GP}_q(m_q, k_q)\,,$$

where the training inputs are denoted by $\mathcal{X}$ and $\mathcal{U}$, respectively. The training outputs are recursively determined by GPDP. A sketch of the GPDP algorithm is given in Algorithm 1. The advantage of modeling the state-value function V_k by $\mathcal{GP}_v$ is that the GP provides a distribution of $V_k(\mathbf{x}_*)$ for *any* state $\mathbf{x}_*$ through (6) and (7). This property is exploited in the computation of the Q-function (3): Due to the generalization property of $\mathcal{GP}_v$, we are not restricted to a finite set of successor states, when we determine $\mathrm{E}[V_{k+1}(f(\mathbf{x}, \mathbf{u}))]$ in line 6. Although we consider a deterministic system, we have to take an expectation—with respect to the latent function V_{k+1}, which is modeled by $\mathcal{GP}_v$. It turns out that $\mathrm{E}[V_{k+1}(f(\mathbf{x}, \mathbf{u}))] = m_v(f(\mathbf{x}, \mathbf{u}))$. The GP model of Q_k in line 7 generalizes the Q-function to continuous-valued control domains. Note that $\mathcal{GP}_q$ models *only* a function of $\mathbf{u}$ since $\mathbf{x}$ is fixed. Therefore, $\min_{\mathbf{u}} Q_k(\mathbf{x}_i, \mathbf{u}) \approx \min_{\mathbf{u}} m_q(\mathbf{u})$, the minimum of the mean function of $\mathcal{GP}_q$. Thus, the minimizing control $\pi_k^*(\mathbf{x}_i)$ in line 8 is not restricted to the finite set $\mathcal{U}$, but can be selected from the continuous-valued domain $\mathbb{R}^{n_u}$. To find $\pi^*(\mathbf{x}_i)$ we have to resort to numerical methods. Although not considered in this paper, the training inputs $\mathcal{X}$ and $\mathcal{U}$ in GPDP can vary at each iteration step k.

Note that for all $\mathbf{x}_i \in \mathcal{X}$ independent GP models for $Q_k(\mathbf{x}_i, \,\cdot\,)$ are used rather than modeling $Q_k(\,\cdot\,, \,\cdot\,)$ in joint state-action space. This idea is largely based on two observations. First, a good model of Q_k in joint state-action space requires substantially more training points and makes standard GP models computationally very expensive. Second, the Q-function can be discontinuous in $\mathbf{x}$ as well as in $\mathbf{u}$. We eliminate one possible source of discontinuity by treating $Q_k(\mathbf{x}_i, \,\cdot\,)$ and $Q_k(\mathbf{x}_j, \,\cdot\,)$ independently.

B. Learning a Closed-Loop Policy

We interpret the state-feedback $\pi^*(\mathcal{X})$ returned by GPDP as noisy measurements of an optimal policy. To generalize these state-feedback values to a continuous-valued, closed-loop optimal policy π^* on the entire state space, we have to solve a regression problem. We suggest to model the latent policy with a GP.

In many dynamic systems, an optimal policy is discontinuous at the boundary of an unknown subset of the state space, especially if the system is underpowered. Using smoothness favoring covariance functions to model this policy is thus inappropriate. Finding a suitable covariance function reflecting our prior beliefs is very hard. We tackle this problem by observing that in applications of control algorithms to real robots smoothness of controls protects the actuators of the system. Therefore, we assume that a close-to-optimal policy is at least piecewise smooth with possible discontinuities at certain states, where the sign of the control signal changes.

Thus, we attempt to model the policy π^* by switching between *two* locally trained GPs. The main idea of this step is depicted in Figure 1. We split the state-feedback $\pi^*(\mathcal{X})$ returned by GPDP into two subsets of training outputs. One GP is trained only on the subset $\pi_+^*(\mathcal{X}) \subset \pi^*(\mathcal{X})$ of positive controls, the other GP uses the remaining set denoted by $\pi_-^*(\mathcal{X})$. We call these GPs $\mathcal{GP}_+$ and $\mathcal{GP}_-$, respectively. Note that the values $\pi^*(\mathcal{X})$ are known from the GPDP algorithm. Both GP models play the role of local experts in the region of their training sets. Since we assume a locally smooth latent close-to-optimal policy, we use smoothness favoring rational quadratic (RQ) kernels for the two locally trained GPs. An RQ kernel can be seen as a scale mixture of SE kernels with different length-scales [5]. After training, it remains to select one local GP model given a new input $\mathbf{x}_*$. In this paper, this decision is made by a binary GP classifier that selects the most likely local GP model. This classifier

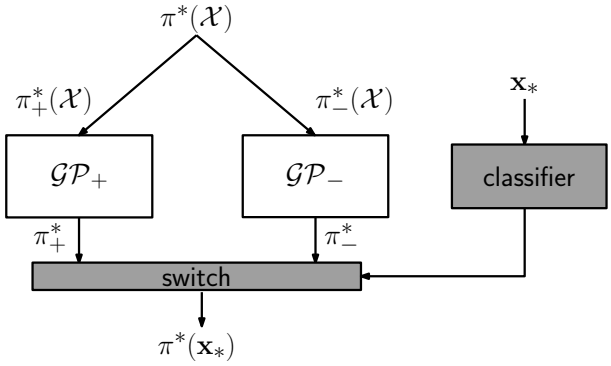

Fig. 1. Policy learning scheme. The optimal state-feedback values $\pi^*(\mathcal{X})$ are split into two groups: positive and negative control signals. Two GPs are trained independently on either of the subsets to guarantee local smoothness. A classifier selects one GP to predict an optimal control for a new input $\mathbf{x}_*$. The resulting policy can be discontinuous along the decision boundary.

plays a similar role as the gating network in a mixture-of-experts setting [11]. We greedily choose the GP model with higher class probability to predict the optimal control to be applied in a state. We always apply the predicted mean of the local GP policy model. Note that convex combination of the predictions of $\mathcal{GP}_+$ and $\mathcal{GP}_-$ according to the corresponding class probabilities will not yield the desired discontinuous policy. Instead, the policy will be smoothed out along the decision boundary, which is not wanted here.

Binary classification maps outcomes of a latent function f into two different classes. In GP classification (GPC) a GP prior is placed over f, which is squashed through a sigmoid function to obtain a prior over the class labels. In contrast to GP regression, the likelihood $p(c_i|f(\mathbf{x}_i))$ in GPC is not Gaussian. The class label of $f(\mathbf{x}_i)$ is $c_i \in \{-1, +1\}$. The integral that yields the posterior distribution of the class labels for new inputs is not analytically computable. Expectation propagation approximates the non-Gaussian likelihood to obtain an approximate Gaussian posterior. We refer to [12], [5] for further details.

We believe that the suggested approach for learning a discontinuous policy using two different GPs is applicable to many dynamic systems and more effective than training a single GP with a problem-specific kernel. Although problem-specific kernels may perform better, they are difficult to determine. Furthermore, local smoothness cannot be guaranteed in many cases.

IV. Experiments

A. Proof of Concept: LQ Problem

We demonstrate that the GPDP algorithm is able to solve the LQ problem with sufficient performance. We chose the linear system $\mathbf{x}_{k+1} = \mathrm{diag}(\begin{bmatrix}0.5 & 1\end{bmatrix})\mathbf{x}_k + \begin{bmatrix}1 & 1\end{bmatrix}^T u_k$ with deterministic squared immediate loss $g(\mathbf{x}_k, u_k) = \mathbf{x}_k^T\mathbf{x}_k + 5u_k^2$. The optimal policy is $\pi^*(\mathbf{x}_k) = -\mathbf{L}\mathbf{x}_k$, where $\mathbf{L} = \begin{bmatrix}0.054259 & 0.333931\end{bmatrix}$. We set the prior mean and covariance functions of $\mathcal{GP}_v$ and $\mathcal{GP}_q$ to $m \equiv 0,\ k = k_{\mathrm{SE}}$. To train the Gaussian processes $\mathcal{GP}_v$ and $\mathcal{GP}_q$ we randomly selected 100 states and 100 control actions. For a set of 100 test points $\mathcal{X}_* \in [-1, 1]^2$ the differences between policy π^*_{GP} of the GPDP controller and policy π^*_{LQ} of the LQ controller are marginal since the normalized mean squared error (NMSE) $\frac{\mathrm{E}[(\pi^*_{\mathrm{GP}}(\mathcal{X}_*) - \pi^*_{\mathrm{LQ}}(\mathcal{X}_*))^2]}{\mathrm{var}(\pi^*_{\mathrm{LQ}}(\mathcal{X}_*))} \approx 0.0008$. The GPDP controller can indeed solve this problem (almost) optimally since there are only marginal differences in the control decisions.

B. Experiment: Underactuated Pendulum Swing Up

In the following, we consider the underpowered pendulum swing up. We assume system dynamics following the ODE

$$\ddot{\varphi}(t) = \frac{-\mu\dot{\varphi}(t) + mgl\sin(\varphi(t)) + u(t)}{ml^2}\,, \qquad (9)$$

where length l, mass m, and the gravitational constant g are given by $l = 1\,\mathrm{m}$, $m = 1\,\mathrm{kg}$, and $\mathrm{g} = 9.81\,\frac{\mathrm{m}}{\mathrm{s}^2}$, respectively. The coefficient of friction is $\mu = 0.05\,\frac{\mathrm{kg\,m}^2}{\mathrm{s}}$. The applied torque is restricted to $u \in [-5, 5]\,\mathrm{Nm}$. Angle and angular velocity are denoted by φ and $\dot{\varphi}$, respectively. The characteristic pendulum frequency is approximately $2\,\mathrm{s}$. Initially, the pendulum is hanging down in state $[\varphi, \dot{\varphi}]^T = [-\pi, 0]^T$. The goal is to swing the pendulum up and to balance it in the inverted position around $[0, 0]^T$. This task has previously been considered a hard problem [13]. Instead of finding a trajectory-based optimal solution as in [13], our goal is to find a globally optimal policy over the entire state space. The pendulum dynamics (9) are temporally discretized according to

$$\mathbf{x}_{k+1} \coloneqq \begin{bmatrix}\varphi_{k+1}\\ \dot{\varphi}_{k+1}\end{bmatrix} = \begin{bmatrix}\varphi_k + \Delta_t\dot{\varphi}_k + \frac{\Delta_t^2}{2}\ddot{\varphi}_k\\ \dot{\varphi}_k + \Delta_t\ddot{\varphi}_k\end{bmatrix}\,, \qquad (10)$$

where $\varphi_k = \varphi(t = k\Delta_t)$ with Δ_t being the time between two samples. The noisy immediate loss g in (8) is

$$g(\mathbf{x}_k, u_k) = 0.1\left(\mathbf{x}_k^T\,\mathrm{diag}(\begin{bmatrix}1 & 0.1\end{bmatrix})\mathbf{x}_k + 0.2\,u_k^2\right) + \varepsilon\,,$$

where the noise standard deviation $\sigma_\varepsilon = 0.001$ has to be accounted for by $\mathcal{GP}_q$. In our case, the sampling rate is $5\,\mathrm{Hz}$. We optimize the undiscounted optimal control problem (1) over 10 time steps. $\mathcal{GP}_q$ is trained on a regular grid of 25 actions $\mathcal{U} \subset [-5, 5]$. To train $\mathcal{GP}_v$ we used a regular grid of 382 states $\mathcal{X} \subset [-\pi, \pi) \times [-7, 7]$. Around the goal state $[0, 0]^T$ we added a regular grid of 16 states with higher resolution. The training set covers the dynamically relevant part of the state space. For both $\mathcal{GP}_q$ and $\mathcal{GP}_v$ we choose the covariance function $k(\mathbf{x}_i, \mathbf{x}_j) \coloneqq k_{\mathrm{SE}}(\mathbf{x}_i, \mathbf{x}_j) + k_{\mathrm{n}}(\mathbf{x}_i, \mathbf{x}_j)$. The noise kernel $k_{\mathrm{n}}(\mathbf{x}_i, \mathbf{x}_j) \coloneqq \sigma_\varepsilon^2\delta_{ij}$ accounts for the noisy immediate loss and smooths out model errors of previous computations. Here, δ_{ij} is the Kronecker delta. At the kth iteration, we define $m_v \coloneqq k =: m_q$ as prior mean functions. This makes states far away from the training set $\mathcal{X}$ unfavorable and penalizes uncertainty in $\mathcal{GP}_q$.

In general, a closed-loop optimal policy for continuous state and control domains cannot be determined. Thus, we rely on DP with state and control space discretization to design a benchmark controller which we compare with the GPDP controller. Here, we used regular grids of approximately 6.2×10^5 states and 125 controls. We consider the DP controller as almost optimal.

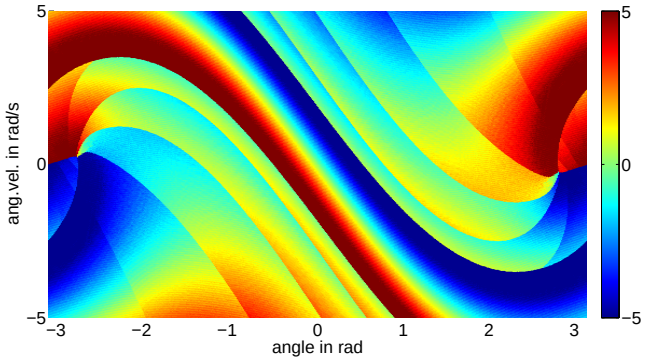

Fig. 2. Optimal policy for a discretized system with 6.2×10^5 states and 125 controls. Discontinuities at the boundary of the central band are caused by the dynamics. Due to temporal discretization, stripes of optimal controls with discontinuous borders appear in the upper right and lower left corners. Red and blue colors show positive and negative controls, respectively.

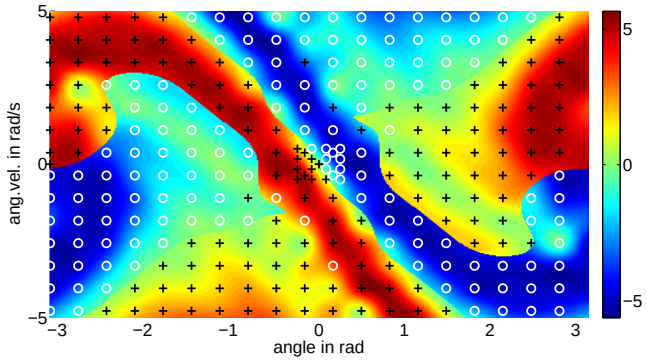

Fig. 3. GP approximation of the learned policy based on 398 training points. The set of optimal control signals is split into two classes on which two GPs are trained independently. The corresponding states (training inputs) are denoted by black crosses and white circles, respectively.

In discrete-time systems, higher temporal sampling rate requires finer spatial discretization. Moreover, especially at the boundaries of the discretized state space, spatial discretization artifacts occur. In contrast, the number of training points in the GPDP algorithm is independent of the temporal sampling rate. Furthermore, the GPDP controller does not suffer from discretization errors since it works in continuous domains

An optimal policy for the underpowered pendulum problem determined by DP is shown in Figure 2. Discontinuities at the boundaries of the diagonal band (upper left to lower right corner) represent states where maximum applicable torque is just not strong enough to bring the pendulum to the inverted position. The controller decides to use torque in opposite direction, to exploit the dynamics of the pendulum, and to bring it to the goal state from the other side. Other discontinuities in Figure 2 are largely due to temporal discretization and decline with higher sampling rates. Figure 2 also shows that a GP model for this policy using smoothness favoring covariance functions, such as SE or RQ kernels is not appropriate if discontinuities at the boundaries of the diagonal band shall be modeled.

The policy model that switches between two GP models as described in Section III-B is given in Figure 3. The black crosses and white circles mark the input locations of the sets $\pi_+(\mathcal{X})$ and $\pi_-(\mathcal{X})$, respectively. The colors describe optimal control decisions. Misclassification is in many cases not a big problem, except along the boundaries of the the diagonal band, where strong discontinuities in the policy appear. More training points in these regions would yield better classification performance, but might also lead to overfitting. Although the range of the controls exceeds the maximally applicable torque, the model of the optimal policy is sufficiently good in most parts of the state space. In simulations we use only the maximally applicable torque.

Starting from the downward position $[-\pi, 0]^T$, we applied nonlinear model predictive control. The trajectories of the system simulation over $5\,\mathrm{s}$ are shown in Figure 4. The first panel of the figure shows the angle, the second panel the

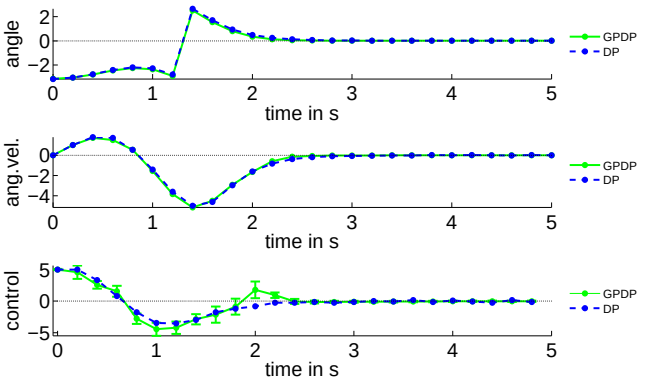

Fig. 4. States and applied control actions of the underpowered pendulum swing up for both controllers. The state trajectories almost coincide, whereas the control signals slightly differ.

angular velocity of the pendulum. The dashed blue curves are caused by the DP controller, the green solid curves are belong to the GPDP controller. The third row describes the applied control signal. The controls applied by the GPDP controller are drawn with an error bar representing the uncertainty in the control decision (twice standard deviation of the GP prediction). Depending on which GP model is used, the uncertainty of the applied control signal depends on the distance of the system state to the training inputs of either $\pi_+(\mathcal{X})$ or $\pi_-(\mathcal{X})$. Both controllers swing the pendulum up and stabilize it in the inverted goal position. The system trajectories almost coincide, the control trajectories differ slightly. The GPDP controller causes with 9.29 slightly more cumulative loss than the DP controller with 9.02.

Note that the GPDP controller finds the solution in continuous-valued domains. Although the GPDP controller is not as good as the optimal benchmark controller, it finds a reasonable tradeoff between performance and generalization in the example considered.

C. Computational and Memory Requirements

GPDP without policy learning scales in $\mathcal{O}^{\text{comp}}_{\text{GPDP}} := \mathcal{O}(|\mathcal{X}||\mathcal{U}|^3 + |\mathcal{X}|^3)$ computations per iteration since GP regression scales cubically in the number of training points.

TABLE I

COMPUTATIONAL DEMANDS AND PERFORMANCES OF GPDP AND DP

GPDP	$\lvert\mathcal{X}\rvert$	$\mathcal{O}^{\text{comp}}_{\text{GPDP}}$	loss	DP	$\lvert\mathcal{X}_{\text{DP}}\rvert$	$\mathcal{O}^{\text{comp}}_{\text{DP}}$	loss
	578	2.0×10^{8}	9.23		6.2×10^{5}	9.8×10^{12}	9.02
	488	1.2×10^{8}	9.24		3.1×10^{5}	1.3×10^{12}	9.05
	400*	7.0×10^{7}	9.55*		2.3×10^{5}	6.1×10^{11}	9.05
	398	6.9×10^{7}	9.29		6.4×10^{3}	1.0×10^{9}	9.34
	200	1.1×10^{7}	9.48		1.5×10^{3}	6.1×10^{7}	11.13

Classical DP for deterministic settings needs $O^{\text{comp}}_{\text{DP}} := \mathcal{O}(|\mathcal{X}_{\text{DP}}|^2|\mathcal{U}_{\text{DP}}|)$ computations. Note that the sets of states $\mathcal{X}_{\text{DP}}$ and controls $\mathcal{U}_{\text{DP}}$ used by DP usually contain substantially more elements than their counterparts in GPDP. Thus, GPDP uses data more efficiently than discretized DP.

In the following, we fix both $\mathcal{U},\ \mathcal{U}_{\text{DP}}$ to 25 controls and only vary the number of states in $\mathcal{X},\ \mathcal{X}_{\text{DP}}$. We compare the performances of DP and GPDP for *one* example trajectory of the underpowered pendulum swing up as well as the corresponding computational requirements. Note that we still consider noisy immediate loss signals, which might slightly decrease the performance of the GPDP controller. The results are given in Table I. For the swing-up trajectory, the best cumulative loss of GPDP in Table I is close to the optimal solution provided by the DP controller although it does not reach it. The computational and memory requirements of GPDP are, however, significantly smaller. If we compare results of similar computational complexity, GPDP with only 398 states outperforms standard DP with 1,500 states. Bigger sets $\mathcal{X}$ in GPDP mainly yield global policies closer to the one from Figure 2. However, increasing the size of $\mathcal{X}$ beyond a certain point tends to overfitting. In Table I, all simulations except the one marked with a * are executed using a regular grid to train $\mathcal{GP}_v$ as described in Section IV-B. The *-entry instead defines $\mathcal{X}$ through *random* samples in the state space. Even in this case, the GPDP controller shows good performance.

Under certain conditions on the dynamics, GPDP can solve the optimal control problem for stochastic systems with little additional computations and no additional memory requirements ($\mathcal{O}(|\mathcal{X}|^2)$). DP is in general no longer applicable because of the $\mathcal{O}(|\mathcal{X}_{\text{DP}}|^2)$ memory required to store a full transition matrix.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced Gaussian process dynamic programming (GPDP). Based on noisy measurements of the immediate loss, Gaussian processes were used to model value functions in the Bellman recursion in the DP algorithm to generalize dynamic programming to continuous-valued state and control domains. This allowed us to avoid discretization problems. GPDP returns an optimal state-feedback for a finite set of states. We suggested to generalize these state-feedback values to a continuous-valued closed-loop policy on the entire state space. To model a close-to-optimal policy that is smooth almost everywhere, we trained two Gaussian processes independently on subsets of the known state-feedback values. Since both Gaussian processes can model the underlying policy well in their training domain, a classifier selects one Gaussian process to predict the optimal control signal for a new query point. Switching between the GP models accounts for discontinuities of the policy along the decision boundary. The application of the concept to a nonlinear problem, the underpowered pendulum swing up, yielded a policy that achieved the task with slightly higher cumulative loss than an almost optimal benchmark controller.

Extending GPDP to stochastic systems within the Gaussian process framework is straightforward. Instead of assuming idealized system dynamics, it is possible to learn the system dynamics based on observations only. Preliminary results strongly motivate the use of GPs for this idea. Optimal placement of support points is also an issue to be dealt with in future. Regression methods need less data points than pure discretization methods. However, especially in high dimensions, data points have to be used efficiently and can be expensive to obtain. Therefore, a criterion has to be formulated to rank and select possible support points for the Gaussian process.

ACKNOWLEDGEMENTS

We thank the reviewers for valuable suggestions. M. P. Deisenroth is supported by the German Research Foundation (DFG) through grant RA 1030/1.

REFERENCES

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed., Athena Scientific, 2005, vol. 1.

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

[3] D. Ormoneit and Ś. Sen, "Kernel-Based Reinforcement Learning," *Machine Learning*, vol. 49, no. 2–3, pp. 161–178, November 2002.

[4] N. Jong and P. Stone, "Kernel-Based Models for Reinforcement Learning," in *ICML Workshop on Kernel Machines and Reinforcement Learning*, Pittsburgh, PA, USA, June 2006.

[5] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[6] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard, "Gaussian Process Model Based Predictive Control," in *Proceedings of the 2004 American Control Conference (ACC 2004)*, Boston, MA, USA, June–July 2004, pp. 2214–2219.

[7] A. Girard, C. E. Rasmussen, J. Quiñonero 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*.The MIT Press, 2003, pp. 529–536.

[8] C. E. Rasmussen and M. Kuss, "Gaussian Processes in Reinforcement Learning," in *Advances in Neural Information Processing Systems 16*. The MIT Press, June 2004, pp. 751–759.

[9] Y. Engel, S. Mannor, and R. Meir, "Reinforcement Learning with Gaussian Processes," in *Proceedings of the 22nd International Conference on Machine Learning (ICML-2005)*, vol. 22, Bonn, Germany, August 2005, pp. 201–208.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[11] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive Mixtures of Local Experts," *Neural Computation*, vol. 3, pp. 79–87, 1991.

[12] T. P. Minka, "A Family of Algorithms for Approximate Bayesian Inference," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, January 2001.

[13] C. G. Atkeson, "Using Local Trajectory Optimizers to Speed up Global Optimization in Dynamic Programming," in *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1994, pp. 503–521.