Neurocomputing 72 (2009) 1508–1524

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Gaussian process dynamic programming

Marc Peter Deisenroth [a,b,*], Carl Edward Rasmussen [a,c], Jan Peters [c,d]

[a] Department of Engineering, University of Cambridge, Cambridge, UK
[b] Faculty of Informatics, Universität Karlsruhe (TH), Germany
[c] Max Planck Institute for Biological Cybernetics, Tübingen, Germany
[d] University of Southern California, Los Angeles, CA, USA

ARTICLE INFO

Available online 12 January 2009

Keywords:
Reinforcement learning
Optimal control
Dynamic programming
Gaussian processes
Bayesian active learning
Policy learning

ABSTRACT

Reinforcement learning (RL) and optimal control of systems with continuous states and actions require approximation techniques in most interesting cases. In this article, we introduce Gaussian process dynamic programming (GPDP), an approximate value function-based RL algorithm. We consider both a classic optimal control problem, where problem-specific prior knowledge is available, and a classic RL problem, where only very general priors can be used. For the classic optimal control problem, GPDP models the unknown value functions with Gaussian processes and generalizes dynamic programming to continuous-valued states and actions. For the RL problem, GPDP starts from a given initial state and explores the state space using Bayesian active learning. To design a fast learner, available data have to be used efficiently. Hence, we propose to learn probabilistic models of the a priori unknown transition dynamics and the value functions on the fly. In both cases, we successfully apply the resulting continuous-valued controllers to the under-actuated pendulum swing up and analyze the performances of the suggested algorithms. It turns out that GPDP uses data very efficiently and can be applied to problems, where classic dynamic programming would be cumbersome.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Reinforcement learning (RL) is based on the principle of experience-based, goal-directed learning. In contrast to supervised learning, where labels are provided from an external supervisor, an RL algorithm must be able to learn from experience collected through interaction with the surrounding world. The objective in RL is to find a strategy, which optimizes a long-term performance measure, such as cumulative reward or cost. RL is similar to the field of optimal control although the fields are traditionally separate. In contrast to optimal control, RL does not necessarily assume problem-specific prior knowledge or an intricate understanding of the world. However, if we call the RL algorithm "controller" and identify actions with the "control signal" we have a one-to-one mapping from RL to optimal control if the surrounding world is fully known. In a general setting, however, an RL algorithm has to explore the world and collect information about it. Since RL is inherently based on collected experience, it provides an intuitive setup for sequential decision-making under uncertainty in autonomous learning.

The RL setup requires to automatically extract information and to *learn* structure from collected data. Learning is important when data sets are very complex or simply too large to find an underlying structure by hand. The learned structure is captured in the form of a statistical model that compactly represents the data. Bayesian data analysis aims to make inferences for quantities about which we wish to learn by using probabilistic models for quantities we observe. The essential characteristic of Bayesian methods is their explicit use of probability theory for quantifying uncertainty in inferences based on statistical data analysis. Without any notion of uncertainty, the RL algorithm would be too confident and claim exact knowledge, which it actually does not have. Representation and incorporation of uncertainties in RL is particularly important in the early stages of learning when the data set is still very sparse. Algorithms based on over-confident models can fail to yield good results due to model bias as reported by Atkeson and Santamaría [2] and Atkeson and Schaal [3]. Hence, it is important to quantify current knowledge appropriately. However, a major drawback of Bayesian methods is that they are computationally costly and the posterior distribution is often not analytically tractable.

Dynamic programming (DP) is a general and efficient method of solving sequential optimization problems under uncertainty. Due to the work of Bellman [4], Howard [19], Kalman [22], and many others, DP became a standard approach to solve optimal control problems. However, only in case of linear systems with

[*] Corresponding author at: Department of Engineering, University of Cambridge, Trumpington Street, Cambridge, UK.
E-mail address: mpd37@cam.ac.uk (M.P. Deisenroth).

0925-2312/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2008.12.019

quadratic cost and Gaussian noise, exact global solutions are known [5]. Similarly, many RL algorithms are based on DP techniques comprising value iteration and policy iteration methods, details of which are given by Sutton and Barto [53], Bertsekas and Tsitsiklis [7]. However, solving a nonlinear optimal control or RL problem for continuous-valued states and actions is challenging and requires approximation techniques in general.

In continuous-valued state and action domains, discretization is commonly used for approximations if required computations are no longer analytically tractable. However, the number of cells in a discretized space does not only depend on the dimensionality and the difficulty of the problem, but also on the time-sampling frequency. The higher the sampling rate, the smaller the size and the larger the number of cells required. Therefore, even low-dimensional problems can be infeasible to solve in discretized spaces. Function approximators address discretization problems and generalize to continuous-valued domains as described for instance by Bertsekas and Tsitsiklis [7] or Sutton and Barto [53]. The key idea is to model the DP value function in a function space rather than representing this function as a table of values at discrete input locations. Parametric function approximators, such as polynomials or radial basis function networks often used for this purpose, but they are only capable of modeling the unknown function within their corresponding model classes. A fundamental problem of parametric function approximators is that the model class is fixed before having observed any data. Often, it is hard to know ahead of time which class of functions will be appropriate. In general, the restriction to a wrong class of functions may result in diverging RL algorithms as shown by Gordon [18], Ormoneit and Sen [38]. Non-parametric regression techniques are generally more flexible than parametric models. “Non-parametric” does not imply that the model is parameter-free, but that the number and nature of the parameters are flexible and not fixed in advance.

Gaussian processes (GPs) combine both flexible non-parametric modeling and tractable Bayesian inference as described by Rasmussen and Williams [48]. The basic idea of non-parametric inference is to use data to infer an unknown quantity based on general prior assumptions. Often, this means using statistical models that are infinite dimensional [55]. Matheron [33] and others introduced GPs to geostatistics decades ago under the name *kriging*. They became popular in the machine learning community in the 1990s through work by Williams and Rasmussen [56] and the thesis by Rasmussen [44]. Recently they got introduced to the control community by Murray-Smith and Sbarbaro [35], Murray-Smith et al. [36] or Kocijan et al. [26], for instance.

GP regression allows for an appropriate uncertainty treatment in RL when approximating unknown functions. For value function and model learning the use of GPs in RL has for instance been discussed for model-free policy iteration by Engel et al. [13,14], for model-based control by Rasmussen and Kuss [47], Murray-Smith and Sbarbaro [35], Rasmussen and Deisenroth [45], and model-based value iteration by Deisenroth et al. [10,11]. Furthermore Ghavamzadeh and Engel [16] discussed GPs in the context of actor-critic methods.

In this article, we introduce and analyze the Gaussian process dynamic programming (GPDP) algorithm. GPDP is a value function-based RL algorithm that generalizes DP to continuous state and action spaces and which belongs to the family of fitted value iteration algorithms [18]. The central idea of GPDP is to utilize non-parametric, Bayesian GP models to describe the value functions in the DP recursion. We will consider both a classic optimal control problem, where much problem-specific prior knowledge is available, and a classic RL problem, where only very general assumptions can be made a priori. In particular, the transition dynamics will be unknown. To solve the RL problem, we will introduce a novel online algorithm that interleaves dynamics learning and value function learning. Moreover, Bayesian active learning is used to deal with the exploration–exploitation trade-off.

The structure of this article is as follows. Section 2 briefly introduces optimal control, RL, and GPs. In Section 3, GPDP is introduced in the context of an optimal control setting, where the transition dynamics are fully known. Furthermore, it will be discussed how a discontinuous, globally optimal policy can be learned if sufficient problem-specific knowledge is available. GPDP will be applied to the illustrative under-actuated pendulum swing up, a nonlinear optimal control problem introduced by Atkeson [1]. In Section 4, we consider a general RL setting, where only very general priors are available. We introduce a very data-efficient, fast learning RL algorithm, which builds probabilistic models of the transition dynamics and value functions on the fly. Bayesian active learning is utilized to efficiently explore the state space. We compare this novel algorithm to the neural fitted Q (NFQ) iteration by Riedmiller [50]. Section 5 summarizes the article.

2. Background

Throughout this article, we consider discrete-time systems

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}, \tag{1}$$

where $\mathbf{x}$ denotes the state, $\mathbf{u}$ the control signal (action), and $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_w)$ a Gaussian distributed noise random variable, where $\mathbf{\Sigma}_w$ is diagonal. Moreover, k is a discrete-time index. The transition function f mapping a state–action pair to a successor state is assumed to evolve smoothly over time.

2.1. Optimal control and RL

Both optimal control and RL aim to find a policy that optimizes a long-term performance measure. A policy π is a mapping from a state space $\mathbb{R}^{n_x}$ into a control space $\mathbb{R}^{n_u}$ that assigns a control signal to each state. In many cases, the performance measure is defined as the expected cumulative cost over a certain time interval. For an initial state $\mathbf{x}_0 \in \mathbb{R}^{n_x}$ and a policy π, the (discounted) expected cumulative cost of a finite N-step optimization horizon is

$$V^{\pi}(\mathbf{x}_0) := \mathrm{E}\left[\gamma^N g_{\mathrm{term}}(\mathbf{x}_N) + \sum_{k=0}^{N-1} \gamma^k g(\mathbf{x}_k, \mathbf{u}_k)\right], \tag{2}$$

where k indexes discrete time. Here, $\mathbf{u} := \pi(\mathbf{x})$ is the control signal assigned by policy π. The function g_{term} is a control-independent terminal cost that incurs at the last time step N. The immediate cost is denoted by $g(\mathbf{x}_k, \mathbf{u}_k)$. The discount factor $\gamma \in (0, 1]$ weights future cost. An optimal policy π^* for the N-step problem minimizes Eq. (2) for any initial state $\mathbf{x}_0$. The associated state-value function V^* satisfies Bellman's equation

$$V^*(\mathbf{x}) = \min_{\mathbf{u}} (g(\mathbf{x}, \mathbf{u}) + \gamma \mathrm{E}_{\mathbf{x}'}[V^*(\mathbf{x}') | \mathbf{x}, \mathbf{u}]) \tag{3}$$

for all states $\mathbf{x}$. 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

$$Q^*(\mathbf{x}, \mathbf{u}) = g(\mathbf{x}, \mathbf{u}) + \gamma \mathrm{E}_{\mathbf{x}'}[V^*(\mathbf{x}') | \mathbf{x}, \mathbf{u}], \tag{4}$$

such that $V^*(\mathbf{x}) = \min_{\mathbf{u}} Q^*(\mathbf{x}, \mathbf{u})$ for all $\mathbf{x}$. In general, finding an optimal policy π^* that leads to Eq. (3) is hard. Assuming time-additive cost and Markovian transitions,[1] the minimal expected

[1] The successor state $\mathbf{x}'$ only depends on the current state–action pair $(\mathbf{x}, \mathbf{u})$.

cumulative cost can be calculated by DP. DP determines the optimal state-value function V^* by the DP recursion

$$V_k^*(\mathbf{x}) = \min_{\mathbf{u}}(g(\mathbf{x},\mathbf{u}) + \gamma \mathrm{E}[V_{k+1}^*(\mathbf{x}')|\mathbf{x},\mathbf{u}]) \tag{5}$$

for all states $\mathbf{x}$ and $k = N-1,\ldots,0$. The state-value function $V_k^*(\mathbf{x})$ is the minimal expected cost over an $N-k$ step optimization horizon starting from state $\mathbf{x}$ at time step k. Analogously to Eq. (5), a recursive approximation of Q^* by Q_k^* can be defined.

The classic DP algorithm is given in Algorithm 1. For known transition dynamics f, a finite set of actions $\mathscr{U}_{\mathrm{DP}}$, and a finite set of states $\mathscr{X}_{\mathrm{DP}}$, DP recursively computes the optimal controls $\pi^*(\mathscr{X}_{\mathrm{DP}})$. Starting from the terminal time N, DP exploits Bellman's optimality principle to determine the value function $V_0^*(X_{\mathrm{DP}})$ and the corresponding optimal controls $\pi_0^*(X_{\mathrm{DP}})$. The value function V_N^* is initialized by the terminal cost g_{term}. The Q^*-values are computed recursively for any state–action pair $(\mathbf{x}_i, \mathbf{u}_j)$ in line 6 of Algorithm 1. For deterministic transition dynamics, the expectation over all successor states in line 6 is not required. The optimal control $\pi_k^*(\mathbf{x}_i)$ of the current recursion step is the minimizing argument of the Q^*-values for a particular state $\mathbf{x}_i$, and the value function $V_k^*(\mathbf{x}_i)$ at $\mathbf{x}_i$ is the corresponding minimum value.

Algorithm 1. Classic DP, known transition dynamics f.

```
1: input: $f, \mathscr{X}_{\mathrm{DP}}, \mathscr{U}_{\mathrm{DP}}$
2: $V_N^*(\mathscr{X}_{\mathrm{DP}}) = g_{\mathrm{term}}(\mathscr{X}_{\mathrm{DP}})$          ▷ terminal cost
3: for $k = N-1$ to 0 do                                   ▷ recursively
4:   for all $\mathbf{x}_i \in \mathscr{X}_{\mathrm{DP}}$ do                          ▷ for all states
5:     for all $\mathbf{u}_j \in \mathscr{U}_{\mathrm{DP}}$ do                        ▷ for all actions
6:       $Q_k^*(\mathbf{x}_i,\mathbf{u}_j) = g(\mathbf{x}_i,\mathbf{u}_j) + \gamma \mathrm{E}_{\mathbf{x}_{k+1}}[V_{k+1}^*(\mathbf{x}_{k+1})|\mathbf{x}_i,\mathbf{u}_j,f]$
7:     end for
8:     $\pi_k^*(\mathbf{x}_i) \in \arg\min_{\mathbf{u}\in\mathscr{U}_{\mathrm{DP}}} Q_k^*(\mathbf{x}_i,\mathbf{u})$
9:     $V_k^*(\mathbf{x}_i) = Q_k^*(\mathbf{x}_i, \pi_k^*(\mathbf{x}_i))$
10:  end for
11: end for
12: return $\pi^*(\mathscr{X}_{\mathrm{DP}}) := \pi_0^*(\mathscr{X}_{\mathrm{DP}})$       ▷ return optimal controls for $\mathscr{X}_{\mathrm{DP}}$
```

In contrast to optimal control, RL usually does not assume a priori known transition dynamics and cost. Hence, general RL algorithms have to treat these quantities as random variables. However, if RL algorithms are applied to a fully known Markov decision process (MDP), the RL problem can be considered equivalent to optimal control. The DP recursion and, therefore, all related algorithms can be used to solve this problem. Both RL and optimal control aim to find a solution to an optimization problem, where the effect of the current decision can be delayed. As an example, we can consider a chess game. The current move will influence all subsequent situations, moves, and decisions, but only at the very end it becomes clear if the match was won or not.

For further details on optimal control, DP, and RL, we refer to the books by Bryson and Ho [8], Bertsekas [5], Bertsekas [6], Bertsekas and Tsitsiklis [7], Sutton and Barto [53].

2.2. Gaussian processes

In the following, a brief introduction to GPs will be given based on the books by MacKay [31] and Rasmussen and Williams [48].

Given a data set $\{\mathbf{X}, \mathbf{y}\}$ consisting of input vectors $\mathbf{x}_i$ and corresponding observations $y_i = h(\mathbf{x}_i) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$, we want to infer a model of the (unknown) function h that generated the data. Here, $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ is the matrix of training inputs, $\mathbf{y} = [y_1, \ldots, y_n]^\top$ is the vector of corresponding training targets (observations). Within a Bayesian framework, the inference of h is described by the posterior probability

$$p(h|\mathbf{X},\mathbf{y}) = \frac{p(\mathbf{y}|h,\mathbf{X})p(h)}{p(\mathbf{y}|\mathbf{X})},$$

where $p(\mathbf{y}|h,\mathbf{X})$ is the likelihood and $p(h)$ is a prior on functions assumed by the model. The term $p(\mathbf{y}|\mathbf{X})$ is called the *evidence* or the *marginal likelihood*. When modeling with GPs, we place a GP prior $p(h)$ directly in the space of functions without the necessity to consider an explicit parameterization of the function h. This prior typically reflects assumptions on the smoothness of h. Similar to a Gaussian distribution, which is fully specified by a mean vector and a covariance matrix, a GP is specified by a mean function $m(\cdot)$ and a covariance function $k(\cdot,\cdot)$, also called a *kernel*.[2] A GP can be considered a distribution over functions. However, regarding a function as an infinitely long vector, all necessary computations for inference and prediction can be broken down to manipulating well-known Gaussian distributions. We write $h \sim \mathcal{GP}(m,k)$ if the latent function h is GP distributed.

Given a GP model of the latent function h, we are interested in predicting function values for an arbitrary input $\mathbf{x}_*$. The predictive (marginal) distribution of the function value $h_* = h(\mathbf{x}_*)$ for a test input $\mathbf{x}_*$ is Gaussian distributed with mean and variance given by

$$\mathrm{E}_h[h_*] = k(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{y}, \tag{6}$$

$$\mathrm{var}_h[h_*] = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2\mathbf{I})^{-1}k(\mathbf{X}, \mathbf{x}_*), \tag{7}$$

where $\mathbf{K} \in \mathbb{R}^{n\times n}$ is the kernel matrix with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

A common covariance function k is the squared exponential (SE)

$$k_{\mathrm{SE}}(\mathbf{x},\mathbf{x}') := \alpha^2 \exp(-\tfrac{1}{2}(\mathbf{x}-\mathbf{x}')^\top \mathbf{\Lambda}^{-1}(\mathbf{x}-\mathbf{x}')) \tag{8}$$

with $\mathbf{\Lambda} = \mathrm{diag}([\ell_1^2, \ldots, \ell_{n_x}^2])$ and ℓ_k, $k = 1,\ldots,n_x$, being the characteristic length-scales. The parameter α^2 describes the variability of the latent function h. The parameters of the covariance function are the hyperparameters of the GP and collected within the vector $\boldsymbol{\theta}$. We optimize them by evidence maximization[3] as recommended by MacKay [30]. The log-evidence is given by

$$\begin{aligned}\log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) &= \log \int p(\mathbf{y}|h(\mathbf{X}),\mathbf{X},\boldsymbol{\theta})p(h(\mathbf{X})|\mathbf{X},\boldsymbol{\theta})\,\mathrm{d}h \\ &= \underbrace{-\frac{1}{2}\mathbf{y}^\top(\mathbf{K}_{\boldsymbol{\theta}} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{y}}_{\text{data fit term}} \underbrace{-\frac{1}{2}\log|(\mathbf{K}_{\boldsymbol{\theta}} + \sigma_\varepsilon^2\mathbf{I})|}_{\text{complexity penalty}} \\ &\quad -\frac{n_x}{2}\log(2\pi).\end{aligned} \tag{9}$$

Here, $h(\mathbf{X}) := [h(\mathbf{x}_1), \ldots, h(\mathbf{x}_n)]$, where n is the number of training points. We made the dependency of $\mathbf{K}$ on the hyperparameters $\boldsymbol{\theta}$ explicit by writing $\mathbf{K}_{\boldsymbol{\theta}}$. Evidence maximization yields a model that (a) rewards the data-fit and (b) rewards simplicity of the model. Hence, it automatically implements Occam's razor.

Maximizing the evidence is a nonlinear, unconstrained optimization problem. Depending on the data set, this can be hard. However, after optimizing the hyperparameters, the GP model can always explain the data although a global optimum has not necessarily been found.

Training a GP requires $\mathcal{O}(n^3)$ operations, where n is the number of training examples. The computational complexity is due to the inversion of the kernel matrix. After training, the predictive mean (6) requires $\mathcal{O}(n)$ operations to compute, the predictive variance (7) requires $\mathcal{O}(n^2)$ operations.

[2] We set the mean function to 0 everywhere, if not stated elsewhere.

[3] Rasmussen and Williams [48] call this marginal likelihood optimization or maximum likelihood type II estimate.

3. Gaussian process dynamic programming

GPDP is a generalization of DP/value iteration to continuous state and action spaces using fully probabilistic GP models [10].

In this section, we consider a discrete-time optimal control problem, where the transition function f in Eq. (1) is exactly known. To determine a solution for continuous-valued state and action spaces, GPDP describes the value functions V_k^* and Q_k^* directly in function space by representing them by fully probabilistic GP models. GP models for this purpose make intuitive sense as they use available data to determine the underlying structure of the value functions, which is often unknown. Moreover, they provide information about the model confidence. Similar to classic DP (see Algorithm 1), we choose finite sets $\mathscr{X}$ of states and $\mathscr{U}$ of actions. However, instead of representing the state and action spaces, these sets are the *support points* (training inputs) for two value function GP models

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

respectively. The training targets (observations) are recursively determined by GPDP itself. A sketch of the GPDP algorithm for known deterministic transition dynamics f is given in Algorithm 2. The advantage of modeling the state-value function V_k^* by $\mathscr{GP}_v$ is that the GP provides a predictive distribution of $V_k^*(\mathbf{x}_*)$ for *any* state $\mathbf{x}_*$ through Eqs. (6) and (7). This property is exploited in the computation of the Q^*-value (line 7): due to the generalization property of $\mathscr{GP}_v$, we are not restricted to a finite set of successor states when determining $\mathrm{E}_V[V_{k+1}^*(f(\mathbf{x}, \mathbf{u}))]$. However, although we consider a deterministic system, we have to take an expectation—with respect to the latent function V_{k+1}^*, which is probabilistically modeled by $\mathscr{GP}_v$. Thus, $\mathrm{E}_V[V_{k+1}^*(f(\mathbf{x}, \mathbf{u}))]$ is simply $m_v(f(\mathbf{x}, \mathbf{u}))$, the predictive mean of $V_k^*(f(\mathbf{x}, \mathbf{u}))$ given by Eq. (6). The GP model of Q_k^* in line 9 generalizes the Q^*-function to continuous-valued action domains. The immediate reward g in line 7 is assumed to be measured with additive independent, Gaussian noise $w_g \sim \mathscr{N}(0, \sigma_g^2)$ with a priori unknown variance σ_g^2. The GP model for Q_k^* takes this variance as additional hyperparameter to be optimized. Note that $\mathscr{GP}_q$ models a function of $\mathbf{u}$ *only* since $\mathbf{x}_i$ 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 $\mathscr{GP}_q$. The minimizing control $\pi_k^*(\mathbf{x}_i)$ in line 10 is not restricted to the finite set $\mathscr{U}$, but can be selected from the continuous-valued control domain $\mathbb{R}^{n_u}$ since for arbitrary controls a predictive distribution of the corresponding Q^*-value is provided by $\mathscr{GP}_q$. To minimize Q_k^* we have to utilize numerical methods.

Algorithm 2. GPDP, known deterministic system dynamics.

```
1: input: $f, \mathscr{X}, \mathscr{U}$
2: $V_N^*(\mathscr{X}) = g_{\mathrm{term}}(\mathscr{X}) + w_g$                          ▷ terminal cost
3: $V_N^*(\cdot) \sim \mathscr{GP}_v$                                                 ▷ GP model for $V_N^*$
4: for $k = N-1$ to 0 do                                                             ▷ recursively
5:     for all $\mathbf{x}_i \in \mathscr{X}$ do                                      ▷ for all support states
6:         for all $\mathbf{u}_j \in \mathscr{U}$ do                                  ▷ for all support actions
7:             $Q_k^*(\mathbf{x}_i, \mathbf{u}_j) = g(\mathbf{x}_i, \mathbf{u}_j) + w_g + \gamma \mathrm{E}_V[V_{k+1}^*(f(\mathbf{x}_i, \mathbf{u}_j))]$
8:         end for
9:         $Q_k^*(\mathbf{x}_i, \cdot) \sim \mathscr{GP}_q$                           ▷ GP model for $Q_k^*$
10:        $\pi_k^*(\mathbf{x}_i) \in \arg\min_{\mathbf{u} \in \mathbb{R}^{n_u}} Q_k^*(\mathbf{x}_i, \mathbf{u})$
11:        $V_k^*(\mathbf{x}_i) = Q_k^*(\mathbf{x}_i, \pi_k^*(\mathbf{x}_i))$
12:    end for
13:    $V_k^*(\cdot) \sim \mathscr{GP}_v$                                             ▷ GP model for $V_k^*$
14: end for
15: return $\mathscr{GP}_v, \mathscr{X}, \pi^*(\mathscr{X}) := \pi_0^*(\mathscr{X})$
```

Note that for all $\mathbf{x}_i \in \mathscr{X}$ 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 three observations. First, we are finally only interested in the values $V_k^*(\mathbf{x}_i)$, the minimal expected cumulative cost at a support point for the V^*-function GP. Therefore, a model of Q_k^* in joint state–action space is not necessary. Second, a good model of Q_k^* in joint state–action space requires substantially more training points and makes standard GP models computationally very expensive. Third, 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.

Summarizing, the generalization of DP to continuous actions is achieved by the Q^*-function model, the generalization to continuous states is achieved by the V^*-function model.

3.1. Computational and memory requirements

GPDP as described in Algorithm 2 requires $\mathscr{O}(|\mathscr{X}||\mathscr{U}|^3 + |\mathscr{X}|^3)$ computations per time step since training a GP scales cubically in the number of training points, see Section 2.2. Classic DP for deterministic settings requires $\mathscr{O}(|\mathscr{X}_{\mathrm{DP}}||\mathscr{U}_{\mathrm{DP}}|)$ computations: the Q^*-value for any state–action pair $(\mathbf{x}_i, \mathbf{u}_j)$ has to be computed. Note that the sets of states $\mathscr{X}_{\mathrm{DP}}$ and actions $\mathscr{U}_{\mathrm{DP}}$ used by DP usually contain substantially more elements than their counterparts in GPDP. Thus, GPDP can use data more efficiently than discretized DP.

In terms of memory requirements, the most demanding part of GPDP is the storage of the inverse kernel matrices $\mathbf{K}_v^{-1}$ and $\mathbf{K}_q^{-1}$, which contain $|\mathscr{X}|^2$ and $|\mathscr{U}|^2$ elements, respectively.

In contrast to classic DP, GPDP is independent of the time-sampling frequency since the set $\mathscr{X}$ contains support points of the GP value function models rather than representations of the state space. Higher time-sampling frequency will require an increase in the number and a decrease in the size of cells in a classic DP setting, where the state space itself is defined by $\mathscr{X}$.

3.2. Policy learning

To learn an optimal, continuous-valued policy on the entire state space, we have to model the policy based on a finite number of evaluations. We regard the policy as a deterministic map from states to actions. Although any function approximator can be used for policy modeling purposes, we approximate the policy with a GP, the *policy GP*.[4]

We interpret the optimal controls $\pi^*(\mathscr{X})$ (line 15 of Algorithm 2) returned by GPDP as noisy measurements of an optimal policy. We assume noisy measurements to account for model errors and the noisy immediate cost function g. To generalize that finite set of optimal controls to a continuous-valued, globally optimal policy π^* on the entire state space, we have to solve a regression problem. The training inputs for the proposed policy GP are the locations $\mathscr{X}$, that is, the training input locations of the value function GP. The training targets are the values $\pi^*(\mathscr{X})$. If we lack problem-specific priors, this general approach is applicable.

Let us consider an example, where this problem-specific prior knowledge is available and discuss a way of learning a discontinuous optimal policy. Discontinuous policies often appear in under-actuated systems. Traditional policy learning methods as discussed by Peters and Schaal [39–41] or standard GP models with smoothness favoring covariance functions, which have been used for instance by Rasmussen and Deisenroth [45], are inappropriate to model discontinuities.

[4] Other function approximators can be employed as well. We use GPs to stay in the same class of function approximators throughout this article.

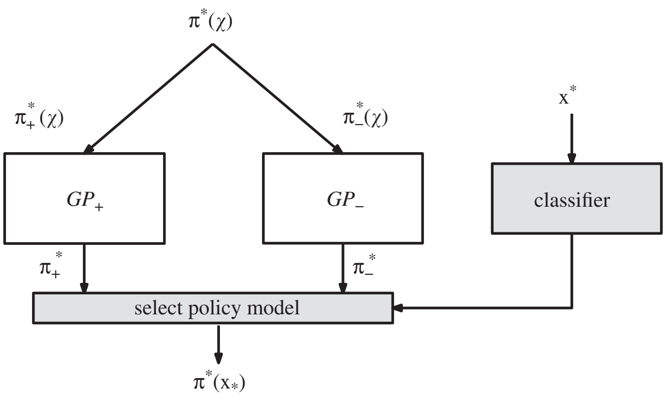


Fig. 1. Learning a discontinuous policy by switching between GP models. The optimal controls $\pi^*(\mathcal{X})$ are split into two groups: positive and negative control signals. Two GPs are trained independently on either of the subsets to guarantee local smoothness. A classifier selects greedily one GP to predict an optimal control for a test input $\mathbf{x}_*$. The resulting policy can be discontinuous along the decision boundary.

In the following, we assume that there exists a near-optimal policy that is piecewise smooth with possible discontinuities at certain states, where the sign of the control signal changes. Due to these considerations, we attempt to model the policy π^* by switching between *two* GPs. The main idea of this step is depicted in Fig. 1. The set of optimal controls $\pi^*(\mathcal{X})$ returned by GPDP is split into two subsets of training targets: controls with positive sign and controls with negative sign. One GP is trained solely on the subset $\pi^*_+(\mathcal{X}) \subset \pi^*(\mathcal{X})$ of positive controls and the corresponding input locations, the other GP uses the remaining set denoted by $\pi^*_-(\mathcal{X})$. As the training inputs of either GP model is restricted to a part of the entire training set, we call them "locally trained". We denote the corresponding GPs by $\mathcal{GP}_+$ 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. After training, it remains to select a single GP model given a test input $\mathbf{x}_*$. In the considered case, this decision is made by a binary (GP) classifier that selects the most likely local GP model to predict the optimal control.[5] The training inputs of the classifier are the states $\mathcal{X}$ and the corresponding targets are the labels "+" or "−", depending on the values $\pi^*(\mathcal{X})$. This classifier plays a similar role as the gating network in a mixture-of-experts setting introduced by Jacobs et al. [20]. In contrast to the work by Jacobs et al. [20], 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 locally trained GP policy model although we obtain distributions over the policies $p(\pi^*_+)$ and $p(\pi^*_-)$, respectively. 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.

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 test inputs is not analytically computable. The expectation propagation (EP) algorithm approximates the non-Gaussian likelihood to obtain an approximate Gaussian posterior. We refer to the work by Minka [34] or the book by Rasmussen and Williams [48] for further details.

Combining GPDP with a policy learning method yields the full RL algorithm (Algorithm 3) that is dealt with in this article. The algorithm determines a continuous-valued (probabilistic) value function model and a continuous-valued policy model.

Algorithm 3. Full RL algorithm.

```
1: $(V^*, \mathcal{X}, \pi^*(\mathcal{X})) = $ GPDP                    ▷ learn value function
2: $\pi^* = $ learn_policy($\mathcal{X}, \pi^*(\mathcal{X})$)                ▷ learn policy
```

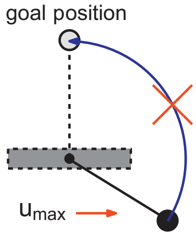
3.3. Evaluations

We analyze GPDP by applying it to a comprehensible, but still challenging, nonlinear control problem, the under-actuated pendulum swing up. The algorithms are implemented using the gpml toolbox from the book by Rasmussen and Williams [48]. At http://mlg.eng.cam.ac.uk/marc/, additional code will be publicly available.

3.3.1. General setup

We consider a discrete-time approximation of the continuous-time pendulum dynamics governed by the ODE

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



where $\mu = 0.05\,\mathrm{kg\,m^2/s}$ is the coefficient of friction, $l = 1\,\mathrm{m}$ is the pendulum length, $m = 1\,\mathrm{kg}$ is the pendulum mass, and $g = 9.81\,\mathrm{m/s^2}$ the gravitational constant. The applied torque is restricted to $u \in [-5, 5]\,\mathrm{N\,m}$ and is not sufficient for a direct swing up. The characteristic pendulum frequency is approximately 0.5 Hz. Angle and angular velocity are denoted by φ and $\dot{\varphi}$, respectively. The control signal is piecewise constant and can be modified every 200 ms. Starting from an arbitrary state, the task is to swing the pendulum up and to balance it in the inverted position around the goal state $[0,0]^\top$. Atkeson and Schaal [3] show that this task is not trivial. Moreover, discretization can become prohibitively expensive despite the low dimensionality as shown by Doya [12]. To avoid discretization, we apply GPDP to work directly in function space and minimize the undiscounted expected total cost (2) over a horizon of 2 s. We choose the saturating immediate cost function g

$$g(\mathbf{x}, u) = 1 - \exp(-\mathbf{x}^\top \mathrm{diag}([1, 0.2])\mathbf{x}) \in [0, 1], \tag{10}$$

which does not penalize the applied action but only the state. The immediate cost (10) is affected by additive Gaussian noise w_g with standard deviation $\sigma_w = 0.001$, which has to be accounted for by $\mathcal{GP}_q$ and is not a priori known to the controller.

For both value function models $\mathcal{GP}_v$ and $\mathcal{GP}_q$ we choose the covariance function

$$k(\mathbf{x}_i, \mathbf{x}_j) := k_{\mathrm{SE}}(\mathbf{x}_i, \mathbf{x}_j) + k_{\mathrm{n}}(\mathbf{x}_i, \mathbf{x}_j),$$

where k_{SE} is the SE kernel defined in Eq. (8). The noise kernel

$$k_{\mathrm{n}}(\mathbf{x}_i, \mathbf{x}_j) := \sigma_\varepsilon^2 \delta_{ij}$$

[5] It is not required that the classifier is a GP classifier. Other binary classifiers, such as SVMs, can be utilized as well.

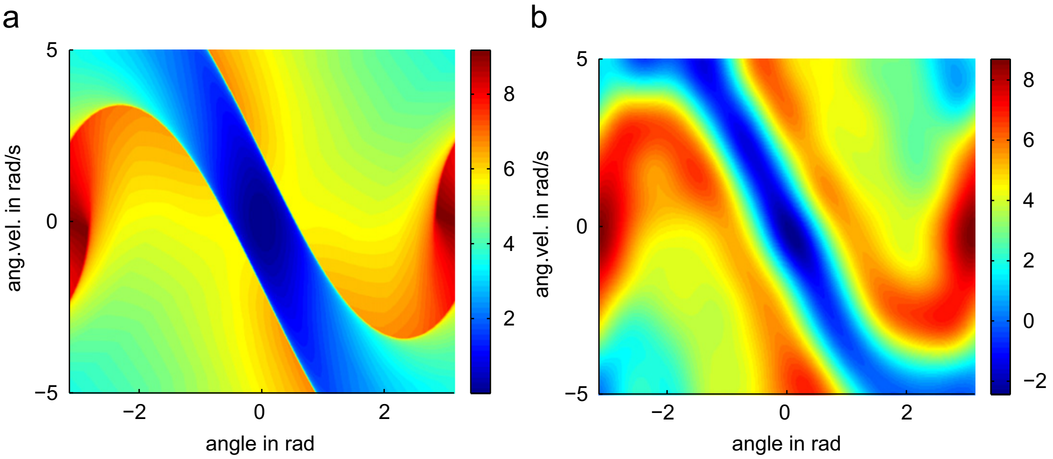


Fig. 2. Optimal and learned value functions. Note that the angle has wrap-around boundary conditions. (a) Optimal DP value function. (b) Mean of value function model (GPDP).

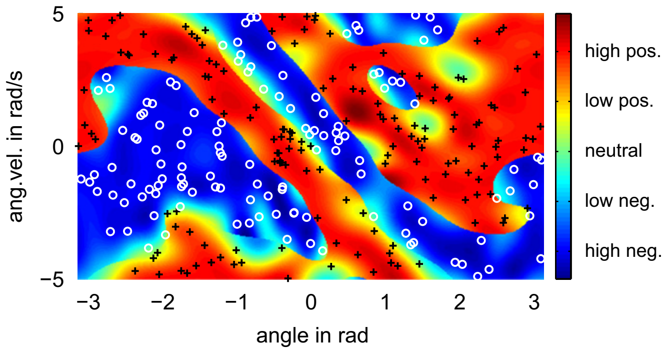


Fig. 3. Mean function of policy model. White circles are the inputs for $\mathcal{GP}_-$, black crosses are the input locations for $\mathcal{GP}_+$. Due to this separation, a GP policy model with discontinuities is determined. The colors encode the strength of the force to be applied (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

smooths model errors of previous computations out. Here, δ_{ij} is the Kronecker delta.[6] We randomly select 400 states[7] as the set of support points $\mathcal{X}$ for $\mathcal{GP}_v$ in the state space hypercube $[-\pi,\pi]^\top$ rad $\times$ $[-7,7]^\top$ rad/s. At the kth iteration, we define the prior mean functions $m_v{:=}k{=:}m_q$ as constant. This makes states far away from the training set $\mathcal{X}$ unfavorable. This setup is reasonable as we assume that the relevant part of the state space is sufficiently covered with support points for the value function GP, $\mathcal{GP}_v$.

For $\mathcal{GP}_q$, a linear grid of 25 actions in the admissible range [5,5] N m defines the training inputs $\mathcal{U}$ of $\mathcal{GP}_q$ for any particular state $\mathbf{x} \in \mathcal{X}$. The training targets are the Q^*-values determined in line 7 of Algorithms 2.

We model the discontinuous policy by switching between two GP models as described in Section 3.2. Since we assume a locally smooth latent near-optimal policy, we use smoothness favoring SE kernels to train the policy models $\mathcal{GP}_+$ and $\mathcal{GP}_-$, respectively.[8] The prior mean functions for $\mathcal{GP}_+$ and $\mathcal{GP}_-$ are set to zero everywhere. Although we do not expect that the positive or negative policies are in average zero, we want the policy to be conservative "in doubt". If the predictive distribution of the optimal control signal has high variance, a conservative policy will not add more energy to the system.

As it is assumed that the deterministic transition dynamics f are a priori known, the considered learning problem almost corresponds to a classic optimal control problem. The only difference is that noisy immediate cost (10) are perceived. To evaluate the quality of the learned policy, we compare it against an optimal solution. In general, an optimal policy for continuous-valued state and control domains cannot be determined. Thus, we rely on classic DP with cumbersome state and control space discretization to design the benchmark controller. Here, we used regular grids of approximately 6.2×10^5 states and 121 possible control values. We consider this DP controller optimal.

3.3.2. *Value function and policy models*

Fig. 2(a) shows the optimal value function determined by DP. The axes define the phase space, that is, angle and angular velocity of the pendulum. Since the pendulum system is under-actuated, the value function is discontinuous around the central diagonal band. The borders are given by states where the applicable torque is just strong enough to perform the swing up, which causes little total cost. In the neighboring state, the pendulum will fall over independent of the torque applied incurring high cumulative cost. The goal state is in the center of the figure at $[0,0]^\top$.

The mean function of the value function model determined by GPDP is given in Fig. 2(b). Although the mean of the value function model $\mathcal{GP}_v$ in the origin is negative, its shape corresponds to the shape of the optimal value function in Fig. 2(a). The discontinuous border is smoothed out, though. Apart from the small negative region in the model, the values are very close to the values of the optimal value function in Fig. 2(a).

The mean of the resulting learned policy is given in Fig. 3. We can model the discontinuous borders of the policy due to the selection of the corresponding locally trained GP as explained in Section 3.2. The white circles in Fig. 3 are the training input locations of $\mathcal{GP}_-$, the black crosses are the training input locations of $\mathcal{GP}_+$. The colors in the plot encode the strengths of the mean predicted torques to be applied. Although some predicted torques can exceed the admissible range of $[-5,5]$ N m, we only apply the maximum admissible torque when interacting with the pendulum system.

3.3.3. *Performance analysis*

Example trajectories of state and applied controls are given in Fig. 4. In the considered particular trajectory, the total cost of the GPDP controller is approximately 9% higher than the total cost

[6] In this article, we restrict ourselves to reporting results with the SE kernel for simplicity reasons. We also analyzed the results for k_q being the Matérn kernel, which gave slightly better results.

[7] We successfully tested the algorithm for 200–600 data points.

[8] Results with the rational quadratic kernel are similar.

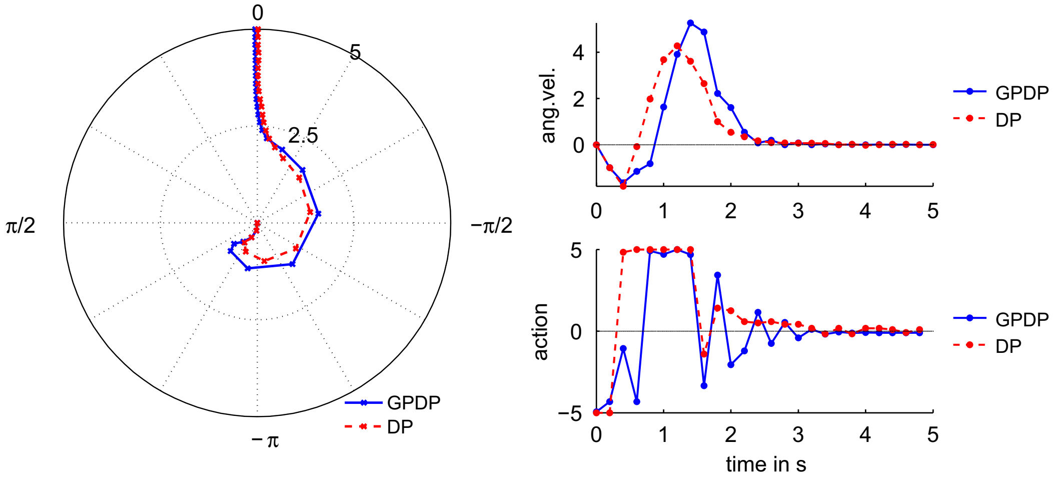


Fig. 4. Example trajectories for the states and the corresponding applied control signals of the under-actuated pendulum swing up for the DP (red, dashed) and GPDP (blue, solid) controllers starting from $[-\pi, 0]^\top$. The left panel is a polar plot of the angle trajectories (in radians) when applying the optimal DP controller (red, dashed) and the GPDP controller (blue, solid). The radius of any graph increases linearly with the time step: at time step zero (initial state $[-\pi, 0]^\top$), the trajectories start in the origin of the figure. Every time step, the radius becomes larger and moves toward the boundary of the polar plot, which it finally reaches at the last time step after 5 s. Both trajectories are close to each other. While the GPDP controller brings the angle more rapidly to the upright position, the DP controller is less aggressive, which is revealed in the angular velocities shown in the right upper panel. The corresponding actions are shown in the right lower panel (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

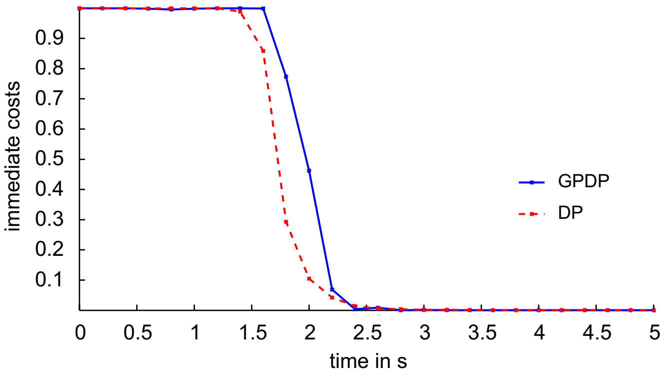


Fig. 5. Immediate cost. Initially, both the DP and the GPDP controller cause full immediate cost. After about 1.5 s, the DP controller starts incurring less cost, whereas the GPDP controller requires another time step to follow. The trajectories of both controllers are approximately cost-free after about 2.4 s as the controllers stabilize the pendulum in the inverted position.

incurring when applying the DP controller. The corresponding incurring immediate cost are shown in Fig. 5.

One thousand initial states $[\varphi_0, \dot{\varphi}_0]^\top \in [-\pi, \pi]^\top$ rad × $[-7, 7]^\top$ rad/s are selected randomly to analyze the global performance of the learned policy. The normalized root mean squared error (NRMSE) is 0.0566 and quantifies the expected error introduced by GPDP compared to the cumbersome optimal DP solution. The average total cost is about 4.6 units for DP and 5.3 units for GPDP. Both controllers are often very similar as for instance shown in Fig. 4, but in rare cases the GPDP controller causes substantially more total cost when it needs an additional pump to swing the pendulum up. However, GPDP *always* solved the task, the maximum total cost incurred was 13.6.

3.3.4. Single GP policy

Thus far, we modeled the policy by switching between locally trained GP models. This problem-specific approach is only applicable if sufficient prior knowledge about a good solution is available. Otherwise, a more general approach is to model the policy with a single GP. The global performance of the single policy is close to the performance we reported for the case of switching between two locally trained GP models. The NRMSE for the single GP policy is 0.0686 (0.0566 for switching GPs), whereas the average cost over 5 s is 5.5 (5.3 for switching GPs). Although the global performances are almost identical, it can happen that the single GP policy performs poorly even when the policy modeled by switching GPs performs well. In particular, this happens if the state trajectory hits a boundary of discontinuity. Such an example is depicted in Fig. 6, where the initial state lies close to such a boundary.

3.4. Discussion

Training $\mathcal{GP}_q$ scales cubically in the number of actions used for training. If the action space cannot easily be covered with training points, subsampling actions is possible to speed up training: assume that the most relevant part of the Q_k^*-function (line 7 of Algorithm 2) is the one close to the optimum and choose those M actions that yield the lowest expected cost in state $\mathbf{x}_i$. These M actions define $\mathcal{U}$ and are the training inputs of $\mathcal{GP}_q$ in Algorithm 2. Then, we obtain more training points in the part of the action space which results in a good approximation performance of the GP model around the optimum of Q_k^*. A similar perspective to this kind of local function approximation is mentioned by Martinez-Cantin et al. [32].

In line 10 of Algorithm 2, we minimize the mean function of $\mathcal{GP}_q$, that is, we do not take the variance of $\mathcal{GP}_q$ into account. Instead of simply minimizing the predictive mean function, it is possible to add a fraction of the predictive variance. This approach will favor actions that yield little expected predictive cost, but will penalize uncertain predictions.

The suggested approach for learning a discontinuous policy by using two different GPs seems 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. However, selecting the switching criterion can vary from case to case. In the considered case, the distinction between positive and negative

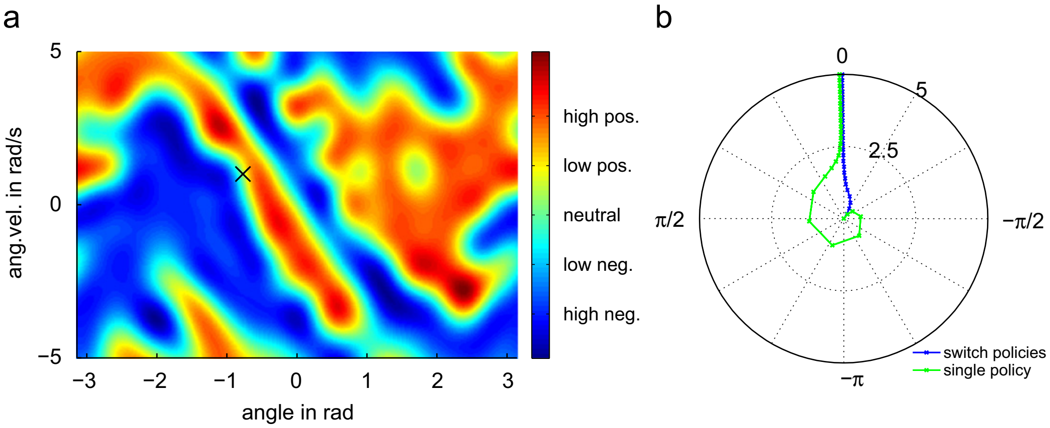


Fig. 6. The effect of smoothing out discontinuities in the policy is displayed: when starting from the state $[-0.77, 1]^\top$, which is close to the boundary where the pendulum falls over, the discontinuous policy (blue) still can go straight toward the target state, whereas the smoothed policy (green) lets the pendulum fall over. (a) Learned policy using a single GP. The initial state (black cross) is located close to the discontinuity, which has been smoothed by single GP policy model. For comparison, see Fig. 3, where the discontinuities are modeled by switching between two GP models. (b) Angle trajectories for controllers using switching GPs (blue) and a single GP (green) to model the policy (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

makes sense for intuitive and practical reasons. A point to be discussed in future is the scalability to high-dimensional inputs.

Finding a globally optimal policy is very difficult in general. Moreover, it requires many data points, particularly in higher dimensions. In practical applications, a globally optimal policy is not required, but rather a policy that can solve a particular task. Thus far, we have placed the support points $\mathcal{X}$ for the value function model $\mathcal{GP}_v$ randomly in the state space. We consider this a suboptimal strategy, which can be highly data-inefficient. In the next section, we will describe how to combine both issues, solving a particular task and using data efficiently.

3.5. Summary

We introduced GPDP. Based on noisy measurements of the immediate cost, GPs were used to model value functions to generalize DP to continuous-valued state and control domains. Modeling the value functions directly in function space allowed us to avoid discretization problems. Moreover, we proposed to learn a continuous-valued optimal policy on the entire state space.

For a particular problem, in which problem-specific prior knowledge was available, we switched between two locally trained GPs to model discontinuities in the policy. The application of the concept to a nonlinear problem, the under-actuated pendulum swing up, yielded a policy that achieved the task with slightly higher cumulative cost than an almost optimal benchmark controller.

4. Online learning

A central issue for RL algorithms is the speed of learning, that is, the number of trials necessary to learn a task. Many learning algorithms require a huge number of trials to succeed. In practice, however, the number of actual trials is very limited due to time or physical constraints. In the following, we discuss an RL algorithm in detail, which aims to speed up learning in a general way.

There are broadly two types of approaches to speed up learning of artificial systems. One approach is to constrain the task in various ways to simplify learning. The issue with this approach is that it is highly problem dependent and relies on an a priori understanding of the characteristics of the task. Alternatively, one can speed up learning by extracting more useful information from available experience. This effect can be achieved by carefully modeling the observations. In a practical application, one would typically combine these two approaches. In the following, we are concerned solely with the second approach: How can we learn as fast as possible, given only very limited prior understanding of a task?

In the sequel, we will generalize the assumptions made in the previous section and assume that the transition dynamics f in Eq. (1) are a priori unknown and that we perceive noisy immediate rewards.[9] The objective is to find an optimal policy leading the system from an initial state to the goal state requiring only a small number of interactions with the real system. This constraint also implies that Monte Carlo sampling, and therefore classical model-free RL algorithms, are often infeasible. Hence, it seems worth building a dynamics model since model-based methods often make better use of available information as described by Bertsekas and Tsitsiklis [7, p. 378]. As discussed by Rasmussen and Deisenroth [45], probabilistic models appropriately quantify knowledge, alleviate model bias, and can lead to very data-efficient solutions.

In the sequel, we will build a probabilistic model of the transition dynamics and incorporate it into the GPDP algorithm. We distinguish between training the model offline or online. Training the model offline, that is, prior to the entire planning algorithm in which an optimal policy is determined, requires either a good cover of the state space or sufficiently good prior knowledge of the task, such that we can restrict the state space to a dynamically relevant part. We followed this approach in our previous work [11]. In this article, we will take a more general approach and train the dynamics model online. With "online" we mean that dynamics model and value function models are being built alternately. Solely based on gathered experience, the idea is to explore a relevant region of the state space automatically while using only general prior assumptions. Solving the described problem within a generalized DP framework demands treatments of the exploration–exploitation tradeoff, online dynamics learning, and one-step ahead predictions. We will address all these issues in this section.

To perform a particular task, we will adapt GPDP (Algorithm 2) such that only a relevant part of the state space will be explored.

[9] In this section, we aim at maximizing rewards instead of minimizing cost. Although both approaches are equivalent in their original form, we prefer rewards in this online setting as they can be intuitively combined with information-based rewards.

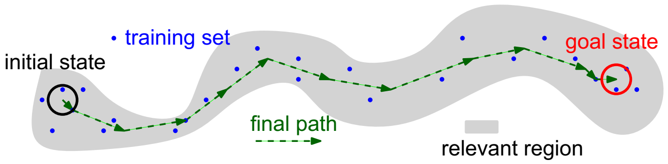


Fig. 7. Starting from an initial state, the algorithm iteratively finds a solution to the RL problem without searching the entire state space, but by placing the training set in relevant regions (shaded area) of the state space only.

Fig. 7 gives an impression how such a solution can be found. Starting from an initial state, training inputs for the involved GP models are placed only in a relevant part of the state space (shaded area). The algorithm finds a solution leading the system through this relevant region to the goal state. GP models of the transition dynamics and the value functions will be built on the fly. The resulting algorithm replaces GPDP in line 1 of Algorithm 3. The policy learning part is not affected. By utilizing Bayesian active learning, we will determine a set of optimal future experiments (interactions with the real system) to use data efficiently.

4.1. Learning the dynamics

We attempt to model short-term transition dynamics based on interactions with the real dynamic system. We assume that the dynamics evolve smoothly over time. Moreover, we implicitly assume time-invariant (stationary) dynamics. We utilize a GP model, the *dynamics GP*, to describe the dynamics $f \sim \mathscr{GP}_f$. For each output dimension i we train a separate GP model

$$x^i_{k+1} - x^i_k \sim \mathscr{GP}(m_f, k_f).$$

This model implies that the output dimensions are conditionally independent given the inputs. Note that the correlation between the state variables is implicitly considered when we observe pairs of states and successor states. The training inputs to the dynamics GP are state–action pairs, the targets are the differences between the successor state and the state in which the action was applied. For any test input $(\mathbf{x}_*, \mathbf{u}_*)$ the predictive distribution of $f(\mathbf{x}_*, \mathbf{u}_*)$ is Gaussian distributed with mean vector $\boldsymbol{\mu}_*$ and covariance matrix $\boldsymbol{\Sigma}_*$. The posterior dynamics GP reveals the remaining uncertainty about the underlying latent function f. For a *deterministic* system, where the noise term $\mathbf{w}$ in Eq. (1) is considered measurement noise, the uncertainty about the latent transition function f tends to zero in the limit of infinite data, and the dynamics GP converges to the deterministic transition function, such that $\mathscr{GP}_f \equiv f$. For a *stochastic* system, the noise term $\mathbf{w}$ in the system equation (1) is process noise. In this case, we obtain a dynamics model $\mathscr{GP}_f$ of the underlying stochastic transition function f that contains *two* sources of uncertainty. First, as in the deterministic case, the uncertainty about the underlying system function itself, and second the uncertainty induced by the process noise. In the limit of infinite data the first source of uncertainty tends to zero, whereas stochasticity due to the process noise $\mathbf{w}$ is always present. This means that only the uncertainty about the model vanishes.

In practice, a deterministic GP model contains only one source of uncertainty as the additive measurement noise can be subtracted from the total uncertainty (measurement noise plus uncertainty about latent function). In the stochastic case, the process noise can never be subtracted as it is part of the transition dynamics.

In the following, we solely consider the case of unknown deterministic transition dynamics with additive measurement noise. Stochastic dynamics with additive process noise can be treated analogously.[10]

Table 1
Solutions to integral (11).

	Known det. f	$\mathscr{GP}_f$
Known V^*	$V^*(f(\mathbf{x},\mathbf{u}))$	$\int V^*(f(\mathbf{x},\mathbf{u}))p(f)\,\mathrm{d}f$
$\mathscr{GP}_v$	$m_v(f(\mathbf{x},\mathbf{u}))$	$\boldsymbol{\beta}^\top \mathbf{l}$

4.2. One-step ahead predictions

Let us revisit the GPDP algorithm (Algorithm 2). In a general RL setting, the deterministic transition dynamics f are no longer known, but rather modeled by the dynamics GP. Assume for a moment that this model is known. The only place where the dynamics come into play is when the Q^*-values are determined. Here, the expected value of V^* at a successor state distribution (line 7 of Algorithm 2),

$$\mathrm{E}_{V,f}[V_{k+1}(\mathbf{x}_{k+1})|\mathbf{x}_i, \mathbf{u}_j, \mathscr{GP}_f] = \iint V(f(\mathbf{x}_i, \mathbf{u}_j))p(V|f)p(f(\mathbf{x}_i, \mathbf{u}_j))\,\mathrm{d}f\,\mathrm{d}V \tag{11}$$

has to be computed for any state–action pair $(\mathbf{x}_i, \mathbf{u}_j) \in \mathscr{X} \times \mathscr{U}$. Both the system function f and the value function V^* are latent and modeled by $\mathscr{GP}_f$ and $\mathscr{GP}_v$, respectively. Explicitly incorporating the uncertainty of the dynamics model in Eq. (11) is important in the context of robust and adaptive control as discussed by Murray-Smith and Sbarbaro [35]. In a Bayesian way, we take the uncertainties about both latent functions into account by averaging over f and V^*. Hence, we have to predict the value of V^* for uncertain inputs $f(\mathbf{x}_i, \mathbf{u}_j)$. We use the Bayesian Monte Carlo method described by Rasmussen and Ghahramani [46] and O'Hagan [37]. In short, the mean and variance of the predictive distribution of $V^*(f(\mathbf{x}_i, \mathbf{u}_j))$ can be computed analytically. The mean is given by

$$\int m_v(f(\mathbf{x}_i, \mathbf{u}_j))p(f(\mathbf{x}_i, \mathbf{u}_j))\,\mathrm{d}f = \boldsymbol{\beta}^\top \mathbf{l} \tag{12}$$

with $\boldsymbol{\beta} := (\mathbf{K} + \sigma_w^2 \mathbf{I})^{-1}\mathbf{y}$ and where

$$l_i = \int k_v(\mathbf{x}_i, f(\mathbf{x}_i, \mathbf{u}_j))p(f(\mathbf{x}_i, \mathbf{u}_j))\,\mathrm{d}f(\mathbf{x}_i, \mathbf{u}_j)$$

is an expectation of $k_v(\mathbf{x}_i, f(\mathbf{x}_i, \mathbf{u}_j))$ with respect to $f(\mathbf{x}_i, \mathbf{u}_j)$. Here, $\mathbf{y}$ are the training targets for $\mathscr{GP}_v$. Further details including the final expression for k_v being the SE covariance function and the corresponding expressions for the predictive variance are given in the paper by Girard et al. [17] and in Appendix A.

Table 1 summarizes four cases of how to solve the integral in Eq. (11) for deterministic dynamics depending on which functions are known. All unknown functions are assumed to be modeled by GPs. To improve readability, we omit the indices i and j in $\mathbf{x}$ and $\mathbf{u}$, respectively. In the first case, we assume that the value function V^* and the dynamics f are deterministic and known. That is, $p(\mathbf{x}'|\mathbf{x}, \mathbf{u}) = \delta(\mathbf{x}' = f(\mathbf{x}, \mathbf{u}))$ is a Dirac delta, and the solution to (11) is simply given by $V^*(f(\mathbf{x}, \mathbf{u}))$. In the second case, we consider a known value function, but unknown dynamics f. The dynamics are modeled by $\mathscr{GP}_f$, and we obtain Gaussian predictions $p(f(\mathbf{x}, \mathbf{u}))$ since $\mathbf{x}' = f(\mathbf{x}, \mathbf{u})$ is Gaussian distributed for any input pair $(\mathbf{x}, \mathbf{u})$. Mean and variance are given by Eqs. (6) and (7), respectively. In combination with nonlinear value functions, the integral in

[10] This is not true for classic DP.

Eq. (11) is only in special cases analytically solvable, even if the value function is exactly known. In the third case, we assume that the dynamics are deterministic and known, but the value function is unknown and modeled by $\mathcal{GP}_v$. This case corresponds to the standard GPDP setting (Algorithm 2) we have proposed in previous work [10]. The expectation with respect to $\mathbf{x}'$ vanishes. However, the expectation has to be taken with respect to the value function V^* to average over the uncertainty of the value function model. Hence, the solution of Eq. (11) is given by $m_v(f(\mathbf{x},\mathbf{u}))$, the evaluation of the mean function of $\mathcal{GP}_v$ at $f(\mathbf{x},\mathbf{u})$. In the fourth case, neither the value function nor the dynamics are exactly known but modeled by $\mathcal{GP}_v$ and $\mathcal{GP}_f$, respectively. Therefore, we have to average over both the uncertainty about the value function and the uncertainty about the dynamics. Due to thesesources of uncertainty, solving the integral (11) corresponds to GP prediction with uncertain inputs $f(\mathbf{x},\mathbf{u})$. The solution is given by Eq. (12).

4.3. Bayesian active learning

It remains to discuss two open problems: How can we learn the transition dynamics online and how do we attack the exploration–exploitation dilemma? We utilize Bayesian active learning (optimal design) to answer both questions.

Active learning can be seen as a strategy for optimal data selection to make learning more efficient. In our case, training data are selected according to a utility function. The utility function often rates outcomes or information gain of an experiment. Before running an actual experiment, these quantities are uncertain. Hence, in Bayesian active learning, the *expected* utility is considered by averaging over possible outcomes.[11] Information-based criteria as proposed by MacKay [29], Krause et al. [27] and Pfingsten [42], for example, or their combination with expected outcomes as discussed by Verdinelli and Kadane [54] and Chaloner and Verdinelli [9] are commonly used to define utility functions. Solely maximizing an expected information gain tends to select states far away from the current state set. MacKay [29] calls this phenomenon the "Achilles' heel" of these methods if the hypotheses space is inappropriate.

To find an optimal policy guiding the system from an initial state to the goal state, we will incorporate Bayesian active learning into GPDP such that only a relevant part of the state space will be explored. GP models of the transition dynamics and the value functions will be built on the fly. A priori it is unclear, which parts of the state space are relevant. Hence, "relevance" is rated by a utility function within a Bayesian active learning framework in which the posterior distributions of the value function model $\mathcal{GP}_v$ will play a central role. This novel online algorithm largely exploits information, which is already computed within GPDP. The combination of active learning and GPDP will be called ALGPDP in the sequel. Instead of a globally, sufficiently accurate value function model, ALGPDP aims to find a locally appropriate value function model in the vicinity of most promising trajectories from the initial states to the goal state.

In RL, the natural setting is that the final objective is to gain both information and high reward. Therefore, we combine the desiderata of expected information gain and expected total rewards to find promising states in the state space that model the value functions well. In a parametric setting, such a utility function has been discussed by Verdinelli and Kadane [54]. We will discuss a non-parametric case in this article.

4.4. ALGPDP

Algorithm 4 describes the entire ALGPDP algorithm. In contrast to GPDP in Algorithm 2, the sets $\mathcal{X}$, are time variant. Therefore, we will denote them by $\mathcal{X}_k$, $k = N, \dots, 0$, in the following, where N is the length of the optimization horizon. ALGPDP starts from a small set of initial input locations $\mathcal{X}_N$. Using Bayesian active learning (line 5), new locations (states) are added to the current set $\mathcal{X}_k$ at any time step k. The sets $\mathcal{X}_k$ serve as training input locations for both the dynamics GP and the value function GP. At each time step, the dynamics model $\mathcal{GP}_f$ is updated (line 6) to incorporate most recent information. Furthermore, the GP models of the dynamics f and the value functions V^* and Q^* are updated. Table 2 gives an overview of the respective training sets, where $\mathbf{x}_i \in \mathcal{X}_k$ and $\mathbf{u}_j \in \mathcal{U}$. Here, $\mathbf{x}'$ denotes an observed successor state of the state–action pair $(\mathbf{x},\mathbf{u})$.

Algorithm 4. Online learning with GPDP.

```
1: train $\mathcal{GP}_f$ around initial states $\mathcal{X}_N$          ▷ initialize dynamics model
2: $V_N^*(\mathcal{X}_N) = g_{\text{term}}(\mathcal{X}_N) + w_g$                  ▷ terminal cost
3: $V_N^*(\cdot) \sim \mathcal{GP}_v$                                     ▷ GP model for $V_N^*$
4: for $k = N-1$ to 0 do                                 ▷ DP recursion (in time)
5:   determine $\mathcal{X}_k$ through Bayesian active learning
6:   update $\mathcal{GP}_f$                                      ▷ GP transition model
7:   for all$\mathbf{x}_i \in \mathcal{X}_k$ do                             ▷ for all support states
8:     for all $\mathbf{u}_j \in \mathcal{U}$ do                            ▷ for all support actions
9:       $Q_k^*(\mathbf{x}_i,\mathbf{u}_j) = g(\mathbf{x}_i,\mathbf{u}_j) + w_g + \gamma \mathrm{E}[V_{k+1}^*(\mathbf{x}_{k+1})|\mathbf{x}_i,\mathbf{u}_j,\mathcal{GP}_f]$
10:    end for
11:    $Q_k^*(\mathbf{x}_i,\cdot) \sim \mathcal{GP}_q$                         ▷ GP model for $Q_k^*$
12:    $\pi_k^*(\mathbf{x}_i) \in \arg\max_{\mathbf{u}\in\mathbb{R}^{nu}} Q_k^*(\mathbf{x}_i,\mathbf{u})$
13:    $V_k^*(\mathbf{x}_i) = Q_k^*(\mathbf{x}_i,\pi_k^*(\mathbf{x}_i))$
14:  end for
15:  $V_k^*(\cdot) \sim \mathcal{GP}_v$                                  ▷ GP model for $V_k^*$
16: end for
17: return $\mathcal{GP}_v, \mathcal{X}, \pi^*(\mathcal{X}_0) := \pi_0^*(\mathcal{X}_0)$
```

4.5. Augmentation of the training sets

ALGPDP starts from a small set of initial input locations $\mathcal{X}_N$. In the following, we define criteria and describe the procedure according to which the training input locations $\mathcal{X}_k$, $k = N-1, \dots, 0$, are found. Let us assume that in each iteration of Algorithm 4, l new states are added to the current input locations $\mathcal{X}_k$. Note that $\mathcal{X}_k$ are the training inputs of the value function GP. The new states are added (line 5 in Algorithm 4) right after training $\mathcal{GP}_v$.

4.5.1. Utility function

Consider a given set $\tilde{\mathcal{X}}$ of possible input locations, which could be added. For efficiency reasons, only the best candidates shall be added to $\mathcal{X}_k$. In RL, we naturally expect from a "good" state $\tilde{\mathbf{x}} \in \tilde{\mathcal{X}}$ to gain both information about the latent value function and high reward. Hence, we choose a utility function U that captures both objectives to rate the quality of candidate states. We aim to find the most promising state $\tilde{\mathbf{x}}^*$ that maximizes the utility function. Due to the probabilistic value function GP model, we consider the expected utility requiring Bayesian averaging. In the context of

Table 2
Training sets of GP models involved in Algorithm 4.

	$\mathcal{GP}_f$	$\mathcal{GP}_v$	$\mathcal{GP}_q$
Training inputs	$(\mathbf{x}_i,\mathbf{u}_i)$	$\mathbf{x}_i$	$\mathbf{u}_j$
Training targets	$\mathbf{x}_i' - \mathbf{x}_i$	$\max_{\mathbf{u}\in\mathbb{R}^{nu}} Q^*(\mathbf{x}_i,\mathbf{u})$	$Q^*(\mathbf{x}_i,\mathbf{u}_j)$

[11] Note that the utility function in this context does not necessarily depend on the RL reward function.

GPDP, we define the expected utility as

$$U(\tilde{\mathbf{x}}):=\rho \mathrm{E}_V[V_k^*(\tilde{\mathbf{x}})|\mathcal{X}_k]+\frac{\beta}{2}\log(\mathrm{var}_V[V_k^*(\tilde{\mathbf{x}})|\mathcal{X}_k]) \tag{13}$$

with weighting factors ρ, β. We explicitly conditioned on the given input locations $\mathcal{X}_k$ on which the current value function has been trained. This utility requires that we have a notion of the distribution of $V_k^*(\tilde{\mathbf{x}})$. Fortunately, the predictive mean and variance

$$\mathrm{E}_V[V_k^*(\tilde{\mathbf{x}})|\mathcal{X}_k]=k_v(\tilde{\mathbf{x}},\mathcal{X}_k)\mathbf{K}_v^{-1}\mathbf{y}_v,$$
$$\mathrm{var}_V[V_k^*(\tilde{\mathbf{x}})|\mathcal{X}_k]=k_v(\tilde{\mathbf{x}},\tilde{\mathbf{x}})-k_v(\tilde{\mathbf{x}},\mathcal{X}_k)\mathbf{K}_v^{-1}k_v(\mathcal{X}_k,\tilde{\mathbf{x}})$$

of $V_k^*(\tilde{\mathbf{x}})$ are directly given by Eqs. (6) and (7), respectively. The utility (13) expresses, how much total reward is expected from $\tilde{\mathbf{x}}$ (first term) and how surprising $V_k^*(\tilde{\mathbf{x}})$ is expected to be given the current training inputs $\mathcal{X}_k$ of the GP model for V_k^* (second term). As described by Chaloner and Verdinelli [9], the second term can be derived from the expected Shannon information (entropy) of the predictive distribution $V_k^*(\tilde{\mathbf{x}})$ or the Kullback–Leibler divergence between the predictive distribution of $V_k^*(\tilde{\mathbf{x}})|\mathcal{X}_k$ and $V_k^*(\mathcal{X}_k)$. The parameters ρ and β assess weight expected reward and expected information gain. A large (positive) value of ρ favors high expected reward, whereas a large value (positive) β favors gaining information based on the predicted variance.[12] Aiming at high expected rewards exploits current knowledge represented and provided by the probabilistic value function model. Gaining information means to explore places with few training points. By adding states with expected high rewards *and* high information gain we lead state trajectories from the initial point to the goal state. Therefore, the parameters ρ,β in Eq. (13) can be considered parameters that control the exploration–exploitation tradeoff.

4.5.2. *Adding multiple states*

Instead of finding only a single promising state $\tilde{\mathbf{x}}^*$, we are interested in the best l states $\tilde{\mathbf{x}}_j^*$, $j=1,\ldots,l$, of the candidate set $\tilde{\mathcal{X}}=\{\tilde{\mathbf{x}}_i : i=1,\ldots,L\}$. A naïve approach is to select all states independently of each other by just taking the best l values of the expected utility (13) when plugging in $\tilde{\mathcal{X}}$. However, we can incorporate cross-information between the candidate states. This approach accounts for the fact that states very close to one another often do not contribute much more information than a single state. To avoid combinatorial explosion in the selection of the best set of l states, we add states sequentially.

We greedily choose the first state $\tilde{\mathbf{x}}_1^* \in \tilde{\mathcal{X}}$ maximizing the expected utility (13). Then, the covariance matrix is augmented according to

$$\mathbf{K}_v:=\begin{bmatrix} \mathbf{K}_v & k_v(\mathcal{X}_k,\tilde{\mathbf{x}}^*) \\ k_v(\tilde{\mathbf{x}}^*,\mathcal{X}_k) & k_v(\tilde{\mathbf{x}}^*,\tilde{\mathbf{x}}^*) \end{bmatrix} \tag{14}$$

with $\tilde{\mathbf{x}}^*=\tilde{\mathbf{x}}_1^*$ and k_v being the covariance function of $\mathcal{GP}_v$. Now, $\mathbf{K}_v$ incorporates information about how $V_k^*(\mathcal{X}_k)$ and $V_k^*(\tilde{\mathbf{x}}_1^*)$ covary. The updated covariance matrix is used to evaluate the expected utility (13), which means to update the predictive variance of $V_k^*(\tilde{\mathbf{x}}_2)$ conditioned on $\mathcal{X}_k$ and $\tilde{\mathbf{x}}_1^*$. Therefore, we explicitly consider cross-covariance information between $V_k^*(\tilde{\mathbf{x}}_1^*)$ and $V_k^*(\tilde{\mathbf{x}}_2)$. The predictive mean of $V_k^*(\tilde{\mathbf{x}}_2)$, the first term in Eq. (13), does not change. Executing this procedure l times determines promising l states $\tilde{\mathbf{x}}_{1,\ldots,l}^* \in \tilde{\mathcal{X}}$. A state $\tilde{\mathbf{x}}_{i+1}$ depends on its expected total reward and its expected information gain conditioned on $\mathcal{X}_k \cup \tilde{\mathbf{x}}_{\leqslant i}^*$. To define the set $\mathcal{X}_{k-1}$, we could use the locations $\tilde{\mathbf{x}}_i^*$, $i=1,\ldots,l$, directly. This approach will cause problems as the states $\tilde{\mathbf{x}}_i^*$ are solely based on *simulation*. If the value function model $\mathcal{GP}_v$ or the transition model $\mathcal{GP}_f$ were totally wrong, it would be possible to add states, which are never dynamically reachable. Hence, we are seeking input locations by *interacting* with the real system.

Thus far, we have discussed how to find promising locations $\tilde{\mathbf{x}}_i^*$ from a set $\tilde{\mathcal{X}}$ of candidates. However, we do not yet know how this set is defined. Moreover, it is not clear yet, how to define the training sets for $\mathcal{GP}_f$ and $\mathcal{GP}_v$ (see Table 2) and how to augment the locations $\mathcal{X}_k$ to obtain $\mathcal{X}_{k-1}$ using the information provided by the promising states $\tilde{\mathbf{x}}_i^*$, which are determined through simulation. We will discuss these issues in the following paragraphs. Note that the locations $\mathcal{X}_k$ serve as training inputs for both the dynamics GP and the value function GP.

4.5.3. *Set of candidate states*

Although it is possible to choose candidate states $\tilde{\mathcal{X}}$ randomly, such selections would be highly inefficient and irregular. Therefore, we take a different approach and exploit the dynamics model for one-step ahead predictions in any recursion within ALGPDP (Algorithm 4), which does not lead us to completely unexplored regions of the state space. Using the dynamics GP, the predicted means of the successor states of the set $\mathcal{X}_k$ (applying the set of actions $\mathcal{U}$ in each of them) are chosen as candidates $\tilde{\mathcal{X}}$. In line 9 of Algorithm 4, these states are denoted by $\mathbf{x}_{k+1}$. Therefore, their predicted state distributions are already known from previous computations.

4.5.4. *Training dynamics and value function models*

In order to train the dynamics model around the initial state (line 1 of Algorithm 4), we observe short trajectories of states starting from the initial state. As we do not have a notion of a good strategy, we may apply actions randomly. The state–action pairs $(\mathbf{x}_i^{\mathrm{init}},\mathbf{u}_i^{\mathrm{init}})$ along the observed trajectories define the training inputs for the dynamics GP, the corresponding successor states $f(\mathbf{x}_i^{\mathrm{init}},\mathbf{u}_i^{\mathrm{init}})$ define the training targets, which can be noisy. We define the set $\mathcal{X}_N:=\{\mathbf{x}_i^{\mathrm{init}}\}_i$ as the training input locations of the initial dynamics GP.

Starting from $\mathcal{X}_N$, we employ Bayesian active learning to augment this set of locations in each iteration of ALGPDP. Assume in the following that the set of input locations $\mathcal{X}_k$ is known. We determine the input locations $\mathcal{X}_{k-1}$ to be employed in the subsequent step of ALGPDP according to the following steps:

1. Determine $\tilde{\mathcal{X}}$, that is, the predicted means of the successor states when starting from $\mathcal{X}_k$ and applying $\mathcal{U}$, $\tilde{\mathcal{X}}:=\mathrm{E}_f[f(\mathcal{X}_k,\mathcal{U})]$. The dynamics GP determines the distribution of the successor states using Eqs. (6) and (7).
2. Bayesian active learning determines the most promising *predicted* states $\tilde{\mathbf{x}}_i^* \in \tilde{\mathcal{X}}$, $i=1,\ldots,l$.
3. Determine l tuples $(\mathbf{x}_i',\mathbf{u}_i') \in \mathcal{X}_k$ such that $\mathrm{E}_f[f(\mathbf{x}_i',\mathbf{u}_i')]=\tilde{\mathbf{x}}_i^* \in \tilde{\mathcal{X}}$. These tuples can be determined by a table look-up since the sets $\mathcal{X}_k$ and $\mathcal{U}$ are finite.
4. We interact with the real system and apply action $\mathbf{u}_i'$ in state $\mathbf{x}_i'$ and *observe* $f(\mathbf{x}_i',\mathbf{u}_i')$. We define $\mathcal{X}_{k-1}:=\mathcal{X}_k \cup \{f(\mathbf{x}_i',\mathbf{u}_i') : i=1,\ldots,l\}$.

Note that we do *not* augment $\mathcal{X}_k$ with the *predicted* states $\tilde{\mathbf{x}}_i^*$, which optimize the utility function (13). Rather, we interact with the real system and apply action $\mathbf{u}_i'$ in state $\mathbf{x}_i$, such that the mean of the successor state is predicted to be $\tilde{\mathbf{x}}_i^*$. We augment $\mathcal{X}_k$ with the corresponding *observation*. Particularly, in the early stages of learning, where not many observations are available, the prediction does not necessarily correspond to the observation. However, the probabilistic dynamics model recognizes and accounts for any

[12] A negative value of β will lead to conservative solutions that avoid solutions with high variance ("pessimism in the face of uncertainty" in contrast to "optimism in the face of uncertainty").

discrepancy between the real observations and the predicted means in the next update.

In line 15 of Algorithm 4, we update the value function model $\mathcal{GP}_v$. The training inputs are the set of states $\mathcal{X}_k$ *and* the goal state.[13] Initially at time step N, the value function equals the terminal reward function g_{term} from Eq. (2), which depends on the state only. In general, the corresponding training targets are defined as the maximum of the Q^*-function evaluated at the locations $\mathcal{X}_k$ and the goal state. The goal state serves as additional training input in the value function model and makes learning more stable and faster since it provides some information about the solution of the task. However, we do not think that this information requires strong prior assumptions: if the rewards are not externally given, the reward function has to be evaluated internally. Note that the maximum immediate reward tells us, where the goal state is.

The utility function (13) is solely optimized for a deterministic set $\tilde{\mathcal{X}}$, which effectively consists of predicted means of successor states. Instead, it is possible to define the utility as a function of the successor state *distribution*. This will require to determine the predictive distribution of V^* with *uncertain* inputs. Mean and variance can be computed analytically and the corresponding expressions for an SE kernel are given in Appendix A. However, when updating the matrix (14), one has to compute the cross-covariances between $V^*(\tilde{\mathbf{x}})$ and $V^*(\tilde{\mathcal{X}}_k)$, which is computationally more involved than computing the corresponding expression for deterministic inputs (which basically is an n-fold evaluation of the kernel). However, computation of the cross-covariance is also analytically tractable. Although a definition of the expected utility based on distributions $p(\tilde{\mathbf{x}})$ will be a clean Bayesian treatment, we do not explicitly discuss this case in this article.

4.6. Computational and memory requirements of ALGPDP

Let us consider the case of unknown (deterministic or stochastic) dynamics first, which are trained *offline*. Apart from training the dynamics GP once, which scales cubically in the number of training points, we have to solve the integral in Eq. (11). Computing a full distribution over the integral can be reformulated as a standard GP prediction, which is quadratic in the number of training points $\mathcal{X}$.[14] However, if we utilize the mean only, the additional computations are $\mathcal{O}(|\mathcal{X}|^2|\mathcal{U}|)$ per time step.

Compared to the case of unknown deterministic transition dynamics, there is *no* additional computational burden for unknown stochastic dynamics. Moreover, no more memory is required to perform necessary computations. DP for stochastic dynamics is often very cumbersome and hardly applicable without approximations because of the $\mathcal{O}(|\mathcal{U}_{\text{DP}}||\mathcal{X}_{\text{DP}}|^2)$ memory required to store a full transition matrix. Moreover, the computational complexity of DP for a stochastic problem is also $O(|\mathcal{U}_{\text{DP}}||\mathcal{X}_{\text{DP}}|^2)$.

Now, let us consider the case of ALGPDP, which trains the transition dynamics and value function models *online*. The extended covariance matrix in Eq. (14) can be inverted in $\mathcal{O}(n^2)$, where n^2 is the number of entries of the previous $\mathbf{K}_v$. Hence, the computational cost of Bayesian active state selection is $\mathcal{O}(|\mathcal{U}|(l|\mathcal{X}_k|^2+(l^2-l)|\mathcal{X}_k|))\in\mathcal{O}(|\mathcal{U}||\mathcal{X}_k|l(l+|\mathcal{X}_k|))$. The dynamics GP can be retrained in $\mathcal{O}((|\mathcal{X}_k|+l)^3)$ since the updated covariance matrix $\mathbf{K}_f$ has to be inverted. The total computational complexity of ALGPDP at time step k is therefore $\mathcal{O}(|\mathcal{U}|(l|\mathcal{X}_k|^2+(l^2-l)|\mathcal{X}_k|)+(|\mathcal{X}_k|+l)^3+|\mathcal{X}_k|^3(1+|\mathcal{U}|)+|\mathcal{U}|^3|\mathcal{X}_k|)\in\mathcal{O}(|\mathcal{U}|(l|\mathcal{X}_k|(l+|\mathcal{X}_k|))+|\mathcal{X}_k|^3(1+|\mathcal{U}|)+|\mathcal{U}|^3|\mathcal{X}_k|)$, which includes training $\mathcal{GP}_f$, $\mathcal{GP}_v$, $\mathcal{GP}_q$, and the evaluation of integral (11) for all successor states of the states $\mathcal{X}_k$ when applying $\mathcal{U}$. Note that $\mathcal{X}_k\subsetneq\mathcal{X}_{k-1}=\mathcal{X}_k\cup\{\tilde{\mathbf{x}}^*_{\leqslant l}\}$ and that standard GPDP in an optimal control setting as discussed in Section 3 utilizes the full set $\mathcal{X}_0$ at *any* time step. Hence, ALGPDP can lead to a remarkable speedup of GPDP.

4.7. Evaluations

We consider the under-actuated pendulum task, which has been introduced in Section 3.3. Instead of minimizing the expected cumulative cost, we now aim to maximize the expected cumulative reward.[15] We will consider the saturating immediate reward function

$$g(\mathbf{x}):=-1+\exp(-\tfrac{1}{2}d(\mathbf{x})^2/a^2)\in[-1,0],\quad a=\tfrac{1}{6}\,\text{m},\tag{15}$$

where

$$d(\mathbf{x})^2=2l^2-2l^2\cos(\varphi),\quad l=1\,\text{m}$$

is the squared distance between the tip of the pendulum and the goal state. Note that the immediate reward (15) solely depends on the angle. In particular, it does not depend on the angular velocity or the control variables. This reward function requires the learning algorithm to discover automatically that a low angular velocity around the goal state is crucial to solve the task. The reward function (15) saturates for angles that deviate more than $17^\circ\approx 0.3$ rad from the goal position.

We maximize the (undiscounted) expected long-term reward over a horizon of 2 s and assume that the dynamics are a priori unknown if not stated elsewhere. The exploration/exploitation parameters in the utility function (13) are set to $\rho:=1,\beta:=2$.[16] The initial state is chosen as $[-\pi,0]^\top$, the goal state is the origin $[0,0]^\top$. The policy is modeled by a single GP instead of two GPs between we can switch to account for discontinuities in the policy. In a general learning approach, we cannot assume that specific prior knowledge is available that describes a properties of a good solution.

4.7.1. Swing up

To learn the transition dynamics around the initial state (line 1 of Algorithm 4), we observe two trajectories of length 2 s, measured and controlled every 200 ms. Initially, we apply actions randomly due to the lack of a good control strategy. The resulting set $\mathcal{X}_N$ consists of 20 states.

To perform the swing up, we use a total of 150 states including the 20 states in $\mathcal{X}_N$ along the initial random trajectories. This means, we augment $\mathcal{X}_k$ by $l=13$ states at each time step to define $\mathcal{X}_{k-1}$. As in Section 3.3, we compared the solution learned by ALGPDP to the optimal DP solution. Fig. 8 shows that a typical solution provided by ALGPDP is close to the quality of the solution of the optimal DP solution. The left panel shows that the angle trajectories are close to each other. Therefore the immediate rewards do not differ much either, which is shown in the right panel. Remember that the reward function (15) is independent of the angular velocity and the control signal. For this particular trajectory, the cumulative reward of ALGPDP is approximately 7% lower than the cumulative reward of the optimal DP solution. Note that due to the reward function (15), only a very small range of angles actually causes rewards significantly deviating from -1.

[13] Riedmiller [50] calls the inclusion of the goal state "hint-to-goal heuristic".

[14] The support points $\mathcal{X}$ are considered time invariant if we train the dynamics offline.

[15] Both objectives are regarded equivalent since a negative reward is the corresponding positive cost.

[16] We did not thoroughly investigate many other parameter settings. However, we observed that the algorithms also work for different values of ρ and β reasonably well.

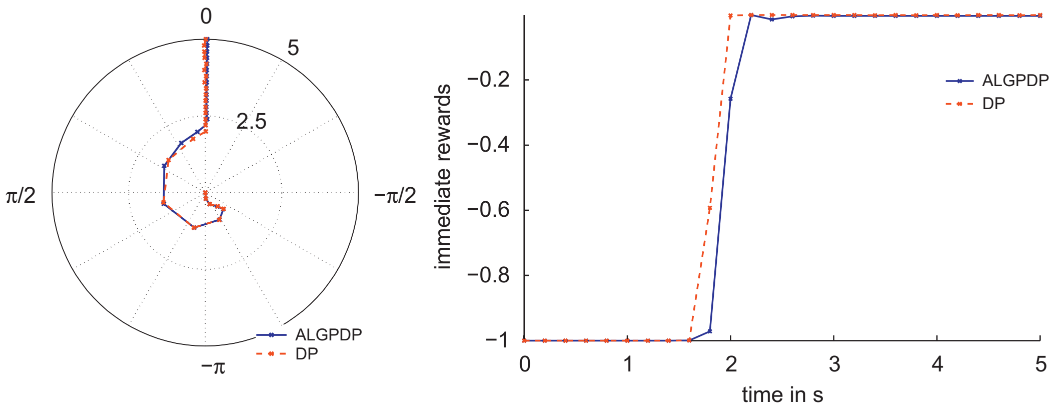


Fig. 8. Trajectories of the angle and immediate rewards when applying optimal policies. The left panel is a polar plot of the angle trajectories (in radians) when applying the optimal DP controller (red, dashed) and the approximate ALGPDP controller (blue, solid). The radius of any graph increases linearly in time: at time step zero (initial state $[-\pi, 0]^\top$), the trajectories start in the origin of the figure. Every time step, the radius becomes larger and moves toward the boundary of the polar plot, which it finally reaches at the last time step after 5 s of simulating the system. Both trajectories are close to each other. The goal state is the upright position, 0 rad. Both controllers move the pendulum rapidly to the goal state in the upright position although the optimal DP controller is slightly faster. The right panel shows the corresponding immediate rewards over time. Initially, the rewards are identical. After 1.8 s they deviate because the DP controller brought the pendulum quicker into the region with higher reward. After 2.2 s both trajectories are in a high-reward zone and do no longer differ noticeably (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

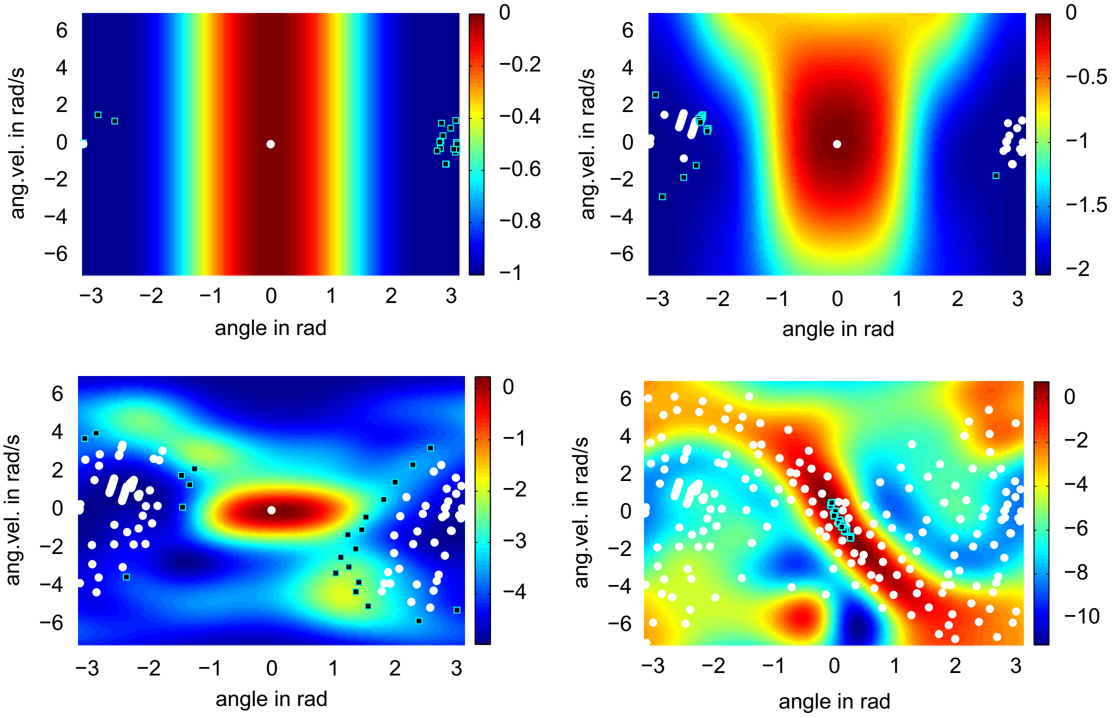


Fig. 9. Means of value function GP after 0, 2, 5, 10 steps of ALGPDP. The GP models of the value function were trained on the input locations marked by the white dots. The upper left panel shows the initial value function model, which was learned with only two input locations: the initial state (left border) and the goal state (center), both in white. The cyan squares are the input locations of the first dynamics model, that is, the random trajectories when starting from the initial state. Note that in all panels the cyan squares are *not* used to train the current value function model, but rather added to the set of states, which serves as training inputs of the next iteration. The upper right panel displays the mean of the value function after two iterations of ALGPDP. The value function is still very flat in the area close to the white dots. Bayesian active learning selects promising locations to fill the relevant part of the state space. Due to its flatness, the expected total reward is not decisive to maximize the utility function. Thus, variance information comes into play and selects locations, where the value function model is very uncertain. Hence, it can happen that the cyan squares are added in uncertain regions in which the expected reward is somewhat lower than elsewhere. The lower left panel shows the mean of the value function GP after five iterations. In this plot, it can already be seen that the recently added states (cyan squares) slowly “move” toward the goal state (white dot in the center), which is the point with highest expected reward. The lower right panel shows the value function model after the last iteration and the full set of 250 input locations. The last states were added close to the goal state, and exploration focuses on high-reward regions close to the goal state. Close to the input locations, which are considered to be the relevant part of the state space, the value function model is sufficiently accurate.

Keep in mind that the optimal DP solution is cumbersome to determine and requires much prior knowledge, computation time, and memory.

Fig. 9 shows a typical evolution of the mean of the probabilistic value function model throughout the iterations of ALGPDP. Starting from the initial random trajectories, input locations are added by using Bayesian active learning. It can be seen that initially the inclusion of new states is based on exploration. The final value function model (lower right plot) is trained with a higher concentration of states around the goal state, which is due

Table 3
Real computation times of ALGPDP on a standard computer.

$\lvert\mathcal{X}_0\rvert = 75$	$\lvert\mathcal{X}_0\rvert = 150$	$\lvert\mathcal{X}_0\rvert = 225$	$\lvert\mathcal{X}_0\rvert = 300$
126 s	256 s	429 s	689 s

to the fact that states in this region are very favorable according to the utility function (13). After finding the high-reward region, the algorithm still explores further until the gap between expected information gain and low reward can no longer be bridged.

ALGPDP can perform the swing up reliably for a size of $\mathcal{X}_0$ of 75–300 states, which corresponds to a total experience (interaction with the system) of less than a minute. The computation times on a standard computer with a 2.4 GHz processor and 2 GB RAM is given in Table 3 for different sizes of $\mathcal{X}_0$. The effective use of data is mainly due to the involved probabilistic models for the dynamics and the value functions. In contrast, Doya [12] solved the task using experience of between 400 and 7000 s meaning that ALGPDP can learn very quickly.

4.7.2. *Comparison to NFQ iteration*

Riedmiller [50] introduced the NFQ iteration as a model-free RL algorithm, which models the Q^*-function by a multi-layer perceptron (MLP). An MLP is a deterministic, non-parametric and is therefore well suited to nonlinear function approximation if the parametric form of the latent function is a priori unknown. However, in contrast to GPs, MLPs usually do not provide confidence about the function model itself. The entire NFQ algorithm is described in Algorithm 5. In the kth iteration, the Q_k^*-function model is trained based on the entire set of transition experiences, $P_{-1}, \ldots, P_k$. The training inputs to the MLP that models Q_k^* are state–action pairs $(\mathbf{x}, \mathbf{u})$, the training targets are the values

$$Q_k^*(\mathbf{x}, \mathbf{u}) = g(\mathbf{x}, \mathbf{u}) + \max_{\mathbf{u}'} \gamma Q_{k-1}^*(\mathbf{x}', \mathbf{u}'),$$

where $\mathbf{x}'$ is the observed successor state of the state–action pair $(\mathbf{x}, \mathbf{u})$ (following an ε-greedy policy). Using the RPROP-algorithm by Riedmiller and Braun [51], the Q^*-function model is updated offline (line 5 of Algorithm 5) to increase data efficiency, which is not given in case of online Q^*-function updates as described by Riedmiller [49]. NFQ collects transition experiences from interactions with the real system, stores them, and reconsiders them for updating the Q^*-function approximator. Riedmiller's NFQ is a general, state-of-the-art RL algorithm and a particular implementation of the fitted Q iteration by Ernst et al. [15].

Algorithm 5. Neural fitted Q iteration.

```
1: init: $P_{-1}$                                          ▷ initialize training pattern
2: $Q_{-1}^* = \mathrm{MLP}(P_{-1})$                         ▷ train initial $Q^*$-function
3: for $k = 0$ to $N$ do
4:     $P_k$ = generate_Pattern                             ▷ collect new data
5:     $Q_k^*$ = Rprop_train($P_{-1}, \ldots, P_k$)          ▷ update $Q^*$-function model
6: end for
7: return $Q^* := Q_N^*$                                     ▷ return final $Q^*$-function model
```

We compare the ALGPDP results from Section 4.7.1 to NFQ with 11 discrete, equidistant actions ranging from −5 to 5 N m.[17] Both algorithms have to solve the swing-up task from scratch, that is, using only very general prior knowledge. The MLP that models the Q^*-function consists of two layers with 20 and 12 units, respectively. The length of an epoch that generates the training pattern P_k is 20 time steps, that is, 8 s. The Q^*-function model requires $N = 64$ iterations to converge. Hence, the final training set consists of 1280 elements, which corresponds to a total experience of approximately 256 s. Note that this NFQ setting aims to find a policy, which is very close to optimal. The reward function used in NFQ is similar to the ALGPDP reward function (15) and does not penalize angular velocity or applied action but solely the distance from a goal. The immediate rewards range from −0.1 to 0. If the pendulum is in a defined goal *region*, maximum reward is gained. A maximum reward region simplifies learning although the reward region in this particular case is very small. In contrast to Bayesian active learning in ALGPDP, NFQ uses an ε-greedy policy to explore the state space.

The optimal actions determined by NFQ quickly bring the pendulum into the upright position and stabilize it there as shown in Fig. 10(a). Compared to ALGPDP (reward −10.25), NFQ (reward −9.66[18]) is even closer to the optimal DP solution (reward −9.60).

With the above setting, the computation time of NFQ on a 2.4 GHz processor is about 1560 s and higher than the computation times of ALGPDP, which are given in Table 3 for different sizes of $\mathcal{X}_0$. Using fewer iterations in NFQ and, therefore, fewer data, still leads to a controller that can solve the swing-up task. For instance, using only 18 (instead of 64) iterations results in a cumulative reward of about −10.1, a solution which corresponds to the quality of the one determined by ALGPDP, which yields a reward of −10.25. The size of the entire NFQ data set decreases to 360 elements, while the required interaction time reduces to 72 s, which is also in the ballpark of the ALGPDP solution requiring less than a minute of interactions. This efficiency is due to the fact that ALGPDP exploits the probabilistic models of the value function and the transition dynamics to explore relevant regions of the state space.

Although the settings of ALGPDP and NFQ were not exactly identical in our evaluations, both algorithms yielded similar results for small data sets. Furthermore, both ALGPDP and NFQ are remarkably more data efficient than the comparable solution to the pendulum swing up by Doya [12].

4.8. *Discussion*

The proposed Bayesian approach of active state selection avoids extreme designs by solely considering states that can be dynamically reached within one time step. Furthermore, it combines an information-based criterion and expected high rewards, the natural choice in RL, to explore the state space. All required mean and variance information (apart from the update of the covariance matrix in Eq. (14)) are directly given by the GP models of the system dynamics, the state-value function V^*, and the state–action value function Q^*. All required Bayesian averaging can be done analytically by exploiting properties of GP models.

We sequentially add new states based on the information provided by the value function GP model. In order to explore the relevant part of the state space, it is necessary to add states every time step. However, already in the setting we discussed in this section, there are states, which do not contribute much to the accuracy of the value function GP (or the dynamics GP). It will be helpful to consider sparse approximations, some of which are discussed in Quiñonero-Candela and Rasmussen [43] to compactly represent the data set. Incorporation of these sparse methods will not be difficult, but remains to future work. In particular, the FITC approximation by Snelson and Ghahramani [52] will be of high interest. Sparse approximations will also be

[17] Roland Hafner and Martin Riedmiller kindly carried the corresponding NFQ experiments out and made the results available.

[18] We applied the reward function (15) to the NFQ trajectory.

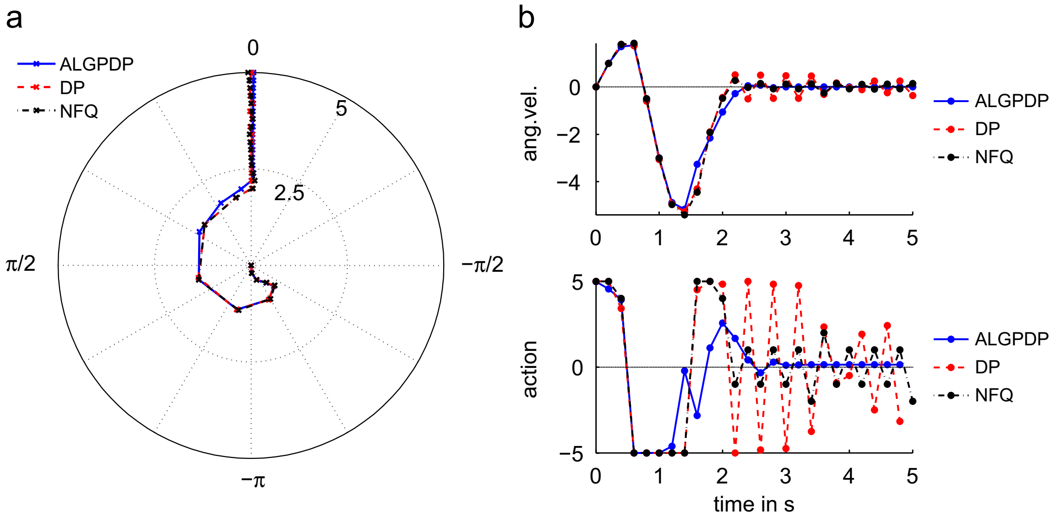


Fig. 10. State and action trajectories for DP, ALGPDP, and NFQ controllers. The trajectories resulting from the NFQ controller are close to the optimal ones determined by the DP controller and slightly outperform the ALGPDP controller. Angles and angular velocities follow the same trend, whereas the applied actions noticeably differ in the stabilization phase. (a) Angle trajectories. (b) Angular velocities and applied actions.

unavoidable if the data sets become remarkably larger. This fact is due to the scaling properties of GP training.

The value function and policy models in ALGPDP depend on the initial trajectories, which are random in our case. Nevertheless, different initializations always led the pendulum to the goal state hinting at the robustness of the method. However, problem-specific prior knowledge can easily be incorporated to improve the models. For example, Ko et al. [24] evaluate a method of combining idealized ODEs describing the system dynamics with GP models for the observations originating from the real system.

The dynamics GP model can be considered an efficient machine learning approach to non-parametric system identification, which models the general input–output behavior. All involved parameters are implicitly determined. A drawback of this method is that using a non-parametric model does usually not yield an interpretable relationship to a mechanical or physical meaning.

If some parameters in system identification cannot be determined with certainty, classic robust control (minimax/$\mathscr{H}_\infty$-control) aims to minimize the worst-case error. This methodology often leads to suboptimal and conservative solutions. Possibly, a fully probabilistic GP model of the system dynamics can be used for robust control as follows. As the GP model reflects uncertainty about the underlying function, it implicitly covers all transition dynamics that explain observed data. By averaging over all these models, we appropriately treat uncertainties and determine a robust controller.

Treatment of noisy measurements in the dynamics learning part is another issue to be dealt with in future. So far, we assumed that we measure the state directly without being squashed through a measurement function. Incorporation of measurement maps demands filter techniques combining predictions and measurements to determine an updated posterior distribution of the hidden state, which is no longer directly accessible. First results in filtering for GP models are already given by Ko et al. [25] and Ko and Fox [23], where GP dynamics and observation models are incorporated in the unscented Kalman filter [21] and the extended Kalman filter.

The proposed ALGPDP algorithm is related to adaptive control and optimal design. Similar ideas have been proposed for instance by Murray-Smith and Sbarbaro [35] and Krause et al. [27].

A major shortcoming of ALGPDP is that it cannot directly be applied to a dynamic system: if we interact with a real dynamic system such as a robot, it is often not possible to experience arbitrary state transitions. A possible adaptation to real-world problems is to experience most promising *trajectories* following the current policy. This approach can basically combine ideas from this article and the paper by Rasmussen and Deisenroth [45].

4.9. Summary

We have introduced a data-efficient model-based Bayesian algorithm for learning control in continuous state and action spaces. GP models of the transition dynamics and the value functions are trained online. We utilize Bayesian active learning to explore the state space and to update the training sets of the current GP models on the fly. The considered utility function rates states according to expected information gain and expected total reward, which seems a natural setting in RL. Our algorithm uses data efficiently, which is important when interacting with the system is expensive.

5. Conclusions

Probabilistic models in artificial learning algorithms can speed up learning noticeably as they quantify uncertainty in experience-based knowledge and alleviate model bias. Hence, they are promising to design data-efficient learning algorithms.

In this article, we introduced Gaussian process dynamic programming (GPDP), a value function-based RL algorithm for continuous-valued state and action spaces. GPDP iteratively models the latent value functions with flexible, non-parametric, probabilistic GPs. In the context of a classic optimal control problem, the under-actuated pendulum swing up, we have shown that GPDP yields a near-optimal solution. However, in this setting, we still required problem-specific knowledge.

To design a general, fast learning algorithm, we extended GPDP, such that a probabilistic dynamics model can be learned online if the transition dynamics are a priori unknown. Furthermore, Bayesian active learning guides exploration and exploitation by sequentially finding states with high expected reward and information gain. This flexibility comes with the price of not modeling the final policy globally, but only locally sufficiently accurate. However, this methodology is useful when only little knowledge about the task and only limited interactions with the real system are available.

We provided experimental evidence that our online algorithm works well on a pendulum swing-up task. The methodology is quite general, relying on GP models, not adapted especially to the pendulum problem. A fairly limited number of points are selected by the active learning algorithm, which enables learning a policy that is very close to the ones found by NFQ iteration, a state-of-the-art model-free RL algorithm, and DP, which uses a very fine discretization with millions of states.

We believe that our algorithm combines aspects, which are crucial to solving more challenging RL problems, such as active online learning and flexible non-parametric modeling. In particular, efficiency in terms of the necessary amount of interaction with the system will often be a limiting factor when applying RL in practice.

Acknowledgments

We are very grateful to Roland Hafner and Martin Riedmiller for performing the neural fitted Q iteration experiments and for valuable discussions. We thank the anonymous reviewers for instructive comments and suggestions. M.P.D. acknowledges support by the German Research Foundation (DFG) through Grant RA 1030/1-3 to C.E.R.

Appendix A. GP prediction with uncertain inputs

In the following, we re-state results from Rasmussen and Ghahramani [46], O'Hagan [37], Girard et al. [17], and Kuss [28] of how to predict with GPs when the test input is uncertain.

Consider the problem of predicting a function value $h(\mathbf{x}_*)$ for an uncertain test input $\mathbf{x}_*$. This problem corresponds to seeking the distribution

$$p(h) = \int p(h(\mathbf{x}_*)|\mathbf{x}_*)p(\mathbf{x}_*)\,\mathrm{d}\mathbf{x}_*. \tag{16}$$

Consider the case, where $h \sim \mathscr{GP}$ with an SE kernel k_h and $\mathbf{x}_* \sim \mathscr{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Let the predictive distribution $p(h(\mathbf{x}_*)|\mathbf{x}_*)$ be given by the standard GP predictive mean and variance, Eqs. (6) and (7), respectively. We can compute the mean v and the variance ψ^2 of the predictive distribution (16) in close form. We approximate the exact predictive distribution with a Gaussian, which possesses the same mean and variance (moment matching). The mean v is given by

$$\begin{aligned} v &= \mathrm{E}_h[\mathrm{E}_{\mathbf{x}_*}[h(\mathbf{x}_*)]] = \mathrm{E}_{\mathbf{x}_*}[\mathrm{E}_h[h(\mathbf{x}_*)]] = \mathrm{E}_{\mathbf{x}_*}[m_h(\mathbf{x}_*)] \\ &= \int m_h(\mathbf{x}_*)p(\mathbf{x}_*)\mathrm{d}\mathbf{x}_* = \boldsymbol{\beta}^\top \mathbf{l} \end{aligned} \tag{17}$$

with $\boldsymbol{\beta} := (\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{y}$ and where

$$\begin{aligned} l_i &= \int k_h(\mathbf{x}_i, \mathbf{x}_*)p(\mathbf{x}_*)\,\mathrm{d}\mathbf{x}_* \\ &= \alpha^2 |\boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu)^\top (\boldsymbol{\Sigma} + \boldsymbol{\Lambda})^{-1}(\mathbf{x}_i - \boldsymbol{\mu})\right) \end{aligned}$$

is an expectation of $k_h(\mathbf{x}_i, \mathbf{x}_*)$ with respect to $\mathbf{x}_*$. Here, $\boldsymbol{\Lambda}$ is a diagonal matrix, whose entries are $\ell_1^2, \ldots, \ell_{n_x}^2$ with $\ell_k, k = 1, \ldots, n_x$, being the characteristic length-scales. Note that the predictive mean v in Eq. (17) depends explicitly on the mean and covariance of the uncertain input $\mathbf{x}_*$. The variance of the predictive distribution $p(h(\mathbf{x}_*))$ is denoted by ψ^2 and given by

$$\begin{aligned} \psi^2 &= \mathrm{E}_{\mathbf{x}_*}[m_h(\mathbf{x}_*)^2] + \mathrm{E}_{\mathbf{x}_*}[\sigma_h^2(\mathbf{x}_*)] - \mathrm{E}_{\mathbf{x}_*}[m_h(\mathbf{x}_*)]^2 \\ &= \boldsymbol{\beta}^\top \mathbf{L}\boldsymbol{\beta} + \alpha^2 - \mathrm{tr}((\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{L}) - v^2 \end{aligned}$$

with

$$L_{ij} = \frac{k_h(\mathbf{x}_i, \boldsymbol{\mu})k_h(\mathbf{x}_j, \boldsymbol{\mu})}{|2\boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|^{1/2}} \exp((\mathbf{z}_{ij} - \boldsymbol{\mu})^\top \left(\boldsymbol{\Sigma} + \frac{1}{2}\boldsymbol{\Lambda}\right)^{-1} \boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1}(\mathbf{z}_{ij} - \boldsymbol{\mu}))$$

and $\mathbf{z}_{ij} := \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j)$. Again, the predictive variance depends explicitly on the mean and the covariance matrix of the uncertain input $\mathbf{x}_*$.

References

[1] C.G. Atkeson, Using local trajectory optimizers to speed up global optimization in dynamic programming, in: J.E. Hanson, S.J. Moody, R.P. Lippmann (Eds.), Advances in Neural Information Processing Systems, vol. 6, Morgan Kaufmann, Los Altos, CA, 1994, pp. 503–521.

[2] C.G. Atkeson, J.C. Santamaría, A comparison of direct and model-based reinforcement learning, in: Proceedings of the International Conference on Robotics and Automation, 1997.

[3] C.G. Atkeson, S. Schaal, Robot learning from demonstration, in: D.H. Fisher Jr (Ed.), Proceedings of the 14th International Conference on Machine Learning, Morgan Kaufmann, Nashville, TN, USA, July 1997, pp. 12–20.

[4] R.E. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, USA, 1957.

[5] D.P. Bertsekas, Dynamic Programming and Optimal Control, Optimization and Computation Series, vol. 1, third ed., Athena Scientific, Belmont, MA, USA, 2005.

[6] D.P. Bertsekas, Dynamic Programming and Optimal Control, Optimization and Computation Series, vol. 2, third ed., Athena Scientific, Belmont, MA, USA, 2007.

[7] D.P. Bertsekas, J.N. Tsitsiklis, Neuro-Dynamic Programming, in: Optimization and Computation, Athena Scientific, Belmont, MA, USA, 1996.

[8] A.E. Bryson, Y.-C. Ho, Applied Optimal Control: Optimization, Estimation, and Control, Hemisphere, New York City, NY, USA, 1975.

[9] K. Chaloner, I. Verdinelli, Bayesian experimental design: a review, Statistical Science 10 (1995) 273–304.

[10] M.P. Deisenroth, J. Peters, C.E. Rasmussen, Approximate dynamic programming with gaussian processes, in: Proceedings of the 2008 American Control Conference, Seattle, WA, USA, June 2008, pp. 4480–4485.

[11] M.P. Deisenroth, C.E. Rasmussen, J. Peters, Model-based reinforcement learning with continuous states and actions, in: Proceedings of the 16th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 2008, pp. 19–24.

[12] K. Doya, Reinforcement learning in continuous time and space, Neural Computation 12 (1) (2000) 219–245.

[13] Y. Engel, S. Mannor, R. Meir, Bayes meets Bellman: the Gaussian process approach to temporal difference learning, in: Proceedings of the 20th International Conference on Machine Learning, Washington, DC, USA, vol. 20, August 2003, pp. 154–161.

[14] Y. Engel, S. Mannor, R. Meir, Reinforcement learning with Gaussian processes, in: Proceedings of the 22nd International Conference on Machine Learning, Bonn, Germany, vol. 22, August 2005, pp. 201–208.

[15] D. Ernst, P. Geurts, L. Wehenkel, Tree-based batch mode reinforcement learning, Journal of Machine Learning Research 6 (2005) 503–556.

[16] M. Ghavamzadeh, Y. Engel, Bayesian policy gradient algorithms, in: B. Schölkopf, J.C. Platt, T. Hoffman (Eds.), Advances in Neural Information Processing Systems, vol. 19, The MIT Press, Cambridge, MA, USA, 2007, pp. 457–464.

[17] A. Girard, C.E. Rasmussen, J. Quiñonero Candela, R. Murray-Smith, Gaussian process priors with uncertain inputs—application to multiple-step ahead time series forecasting, in: S. Becker, S. Thrun, K. Obermayer (Eds.), Advances in Neural Information Processing Systems, vol. 15, The MIT Press, Cambridge, MA, USA, 2003, pp. 529–536.

[18] G.J. Gordon, Stable function approximation in dynamic programming, in: A. Prieditis, S. Russell (Eds.), Proceedings of the 12th International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA, USA, 1995, pp. 261–268.

[19] R.A. Howard, Dynamic Programming and Markov Processes, The MIT Press, Cambridge, MA, USA, 1960.

[20] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, G.E. Hinton, Adaptive mixtures of local experts, Neural Computation 3 (1991) 79–87.

[21] S.J. Julier, J.K. Uhlmann, Unscented filtering and nonlinear estimation, IEEE Review 92 (3) (2004) 401–422.

[22] R.E. Kalman, A new approach to linear filtering and prediction problems, Transactions of the ASME—Journal of Basic Engineering 82 (Series D) (1960) 35–45.

[23] J. Ko, D. Fox, GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models, in: Proceedings of the 2008 IEEE/RSJ

International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 2008, pp. 3471–3476.

[24] J. Ko, D.J. Klein, D. Fox, D. Haehnel, Gaussian processes and reinforcement learning for identification and control of an autonomous blimp, in: Proceedings of the International Conference on Robotics and Automation, Rome, Italy, April 2007, pp. 742–747.

[25] J. Ko, D.J. Klein, D. Fox, D. Haehnel, GP-UKF: unscented Kalman filters with Gaussian process prediction and observation models, in: Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, CA, USA, October 2007, pp. 1901–1907.

[26] J. Kocijan, R. Murray-Smith, C.E. Rasmussen, B. Likar, Predictive control with Gaussian process models, in: B. Zajc, M. Tkalčič (Eds.), Proceedings of IEEE Region 8 Eurocon 2003: Computer as a Tool, Piscataway, NJ, USA, September 2003, pp. 352–356.

[27] A. Krause, A. Singh, C. Guestrin, Near-optimal sensor placements in Gaussian processes: theory, efficient algorithms and empirical studies, Journal of Machine Learning Research 9 (2008) 235–284.

[28] M. Kuss, Gaussian process models for robust regression, classification, and reinforcement learning, Ph.D. Thesis, Technische Universität Darmstadt, Germany, February 2006.

[29] D.J.C. MacKay, Information-based objective functions for active data selection, Neural Computation 4 (1992) 590–604.

[30] D.J.C. MacKay, Comparison of approximate methods for handling hyperparameters, Neural Computation 11 (5) (1999) 1035–1068.

[31] D.J.C. MacKay, Information Theory, Inference, and Learning Algorithms, Cambridge University Press, The Edinburgh Building, Cambridge, UK, 2003.

[32] R. Martinez-Cantin, N. de Freitas, A. Doucet, J. Castellanos, Active policy learning for robot planning and exploration under uncertainty, in: Proceedings of Robotics: Science and Systems III, Atlanta, GA, USA, June 2007.

[33] G. Matheron, The intrinsic random functions and their applications, Advances in Applied Probability 5 (1973) 439–468.

[34] T.P. Minka, A family of algorithms for approximate Bayesian inference, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, January 2001.

[35] R. Murray-Smith, D. Sbarbaro, Nonlinear adaptive control using non-parametric Gaussian process prior models, in: Proceedings of the 15th IFAC World Congress, vol. 15, Academic Press, Barcelona, Spain, July 2002.

[36] R. Murray-Smith, D. Sbarbaro, C.E. Rasmussen, A. Girard, Adaptive, cautious, predictive control with Gaussian process priors, in: 13th IFAC Symposium on System Identification, Rotterdam, Netherlands, August 2003.

[37] A. O'Hagan, Bayes–Hermite quadrature, Journal of Statistical Planning and Inference 29 (1991) 245–260.

[38] D. Ormoneit, Ś Sen, Kernel-based reinforcement learning, Machine Learning 49 (2–3) (2002) 161–178.

[39] J. Peters, S. Schaal, Learning to control in operational space, The International Journal of Robotics Research 27 (2) (2008) 197–212.

[40] J. Peters, S. Schaal, Natural actor-critic, Neurocomputing 71 (7–9) (2008) 1180–1190.

[41] J. Peters, S. Schaal, Reinforcement learning of motor skills with policy gradients, Neural Networks 21 (2008) 682–697.

[42] T. Pfingsten, Bayesian active learning for sensitivity analysis, in: Proceedings of the 17th European Conference on Machine Learning, September 2006, pp. 353–364.

[43] J. Quiñonero-Candela, C.E. Rasmussen, A unifying view of sparse approximate Gaussian process regression, Journal of Machine Learning Research 6 (2) (2005) 1939–1960.

[44] C.E. Rasmussen, Evaluation of Gaussian processes and other methods for non-linear regression, Ph.D. Thesis, Department of Computer Science, University of Toronto, 1996.

[45] C.E. Rasmussen, M.P. Deisenroth, Probabilistic inference for fast learning in control, in: S. Girgin, M. Loth, R. Munos, P. Preux, D. Ryabko (Eds.), Recent Advances in Reinforcement Learning, Lecture Notes on Computer Science, vol. 5323, Springer, Berlin, November 2008, pp. 229–242.

[46] C.E. Rasmussen, Z. Ghahramani, Bayesian Monte Carlo, in: S. Becker, S. Thrun, K. Obermayer (Eds.), Advances in Neural Information Processing Systems, vol. 15, The MIT Press, Cambridge, MA, USA, 2003, pp. 489–496.

[47] C.E. Rasmussen, M. Kuss, Gaussian processes in reinforcement learning, in: S. Thrun, L.K. Saul, B. Schölkopf (Eds.), Advances in Neural Information Processing Systems, vol. 16, The MIT Press, Cambridge, MA, USA, 2004, pp. 751–759.

[48] C.E. Rasmussen, C.K.I. Williams, Gaussian processes for machine learning, in: Adaptive Computation and Machine Learning, The MIT Press, Cambridge, MA, USA, 2006 URL ⟨http://www.gaussianprocess.org/gpml/⟩.

[49] M. Riedmiller, Concepts and facilities of a neural reinforcement learning control architecture for technical process control, Neural Computation and Application 8 (2000) 323–338.

[50] M. Riedmiller, Neural Fitted Q iteration—first experiences with a data efficient neural reinforcement learning method, in: Proceedings of the 16th European Conference on Machine Learning, Porto, Portugal, 2005.

[51] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning the RPROP algorithm, in: Proceedings of the IEEE International Conference on Neural Networks, 1993, pp. 586–591.

[52] E. Snelson, Z. Ghahramani, Sparse Gaussian processes using pseudo-inputs, in: Y. Weiss, B. Schölkopf, J.C. Platt (Eds.), Advances in Neural Information Processing Systems, vol. 18, The MIT Press, Cambridge, MA, USA, 2006, pp. 1257–1264.

[53] R.S. Sutton, A.G. Barto, Reinforcement Learning An Introduction, in: Adaptive Computation and Machine Learning, The MIT Press, Cambridge, MA, USA, 1998.

[54] I. Verdinelli, J.B. Kadane, Bayesian designs for maximizing information and outcome, Journal of the American Statistical Association 87 (418) (1992) 510–515.

[55] L. Wasserman, All of Nonparametric Statistics. Springer Texts in Statistics, Springer Science+Business Media, Inc., New York, NY, USA, 2006.

[56] C.K.I. Williams, C.E. Rasmussen, Gaussian processes for regression, in: D.S. Touretzky, M.C. Mozer, M.E. Hasselmo (Eds.), Advances in Neural Processing Systems, vol. 8, The MIT Press, Cambridge, MA, USA, 1996, pp. 598–604.

Marc Peter Deisenroth is a Ph.D. candidate at Universität Karlsruhe (TH), Germany, while being visiting graduate student at the Computational and Biological Learning Lab at the Department of Engineering, University of Cambridge, UK. He graduated from Universität Karlsruhe (TH) in August 2006 with a German Masters degree in Informatics. From October 2006 to September 2007, he has been a graduate research assistant at the Max Planck Institute for Biological Cybernetics in Tübingen, Germany. He has been a visiting researcher at Osaka University, Japan, in 2006 and at Kanazawa University, Japan, in 2004. His research interests include Bayesian inference, reinforcement learning, optimal and nonlinear control.

Carl Edward Rasmussen is a lecturer in the Computational and Biological Learning Lab at the Department of Engineering, University of Cambridge and an adjunct research scientist at the Max Planck Institute for Biological Cybernetics, Tübingen, Germany. His main research interests are Bayesian inference and machine learning. He received his Masters in Engineering from the Technical University of Denmark and his Ph.D. in Computer Science from the University of Toronto in 1996. Since then he has been a post doc at the Technical University of Denmark, a senior research fellow at the Gatsby Computational Neuroscience Unit at University College London from 2000 to 2002, and a junior research group leader at the Max Planck Institute for Biological Cybernetics in Tübingen, Germany, from 2002 to 2007.

Jan Peters heads the Robot Learning Lab (RoLL) at the Max Planck Institute for Biological Cybernetics (MPI) in Tübingen, Germany, while being an invited researcher at the Computational Learning and Motor Control Lab at the University of Southern California (USC). Before joining MPI, he graduated from University of Southern California with a Ph.D. in Computer Science in March 2007. Jan Peters studied Electrical Engineering, Computer Science and Mechanical Engineering. He holds two German M.S. degrees in Informatics and in Electrical Engineering (from Hagen University and Munich University of Technology) and two M.S. degrees in Computer Science and Mechanical Engineering from USC. During his graduate studies, Jan Peters has been a visiting researcher at the Department of Robotics at the German Aerospace Research Center (DLR) in Oberpfaffenhofen, Germany, at Siemens Advanced Engineering (SAE) in Singapore, at the National University of Singapore (NUS), and at the Department of Humanoid Robotics and Computational Neuroscience at the Advanced Telecommunication Research (ATR) Center in Kyoto, Japan. His research interests include robotics, nonlinear control, machine learning, reinforcement learning, and motor skill learning.