2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

Adaptive Autonomous Control using Online Value Iteration with Gaussian Processes

Axel Rottmann Wolfram Burgard

Abstract— **In this paper, we present a novel approach to controlling a robotic system online from scratch based on the reinforcement learning principle. In contrast to other approaches, our method learns the system dynamics and the value function separately, which permits to identify the individual characteristics and is, therefore, easily adaptable to changing conditions. The major problem in the context of learning control policies lies in high-dimensional state and action spaces, that needs to be explored in order to identify the optimal policy. In this paper, we propose an approach that learns the system dynamics and the value function in an alternating fashion based on Gaussian process models. Additionally, to reduce computation time and to make the system applicable to online learning, we present an efficient sparsification method. In experiments carried out with a real miniature blimp we demonstrate that our approach can learn height control online. Further results obtained with an inverted pendulum show that our method requires less data to achieve the same performance as an off-line learning approach.**

I. Introduction

Optimal control is one of the fundamental tasks in the field of robotics and engineering and a variety of methods to solve this task have been presented. A common framework is dynamic programming and most existing approaches use a discrete approximation of the state and control space. This, however, can lead to discretization errors affected by the relation between the time, state, and control representations [1]. The system dynamics, mostly in terms of ordinary differential equations (ODE), provide a basis for dynamic programming. As they typically depend on the current properties of the environment, it is hard to determine policies that appropriately capture all these potential circumstances. Also assuming stationary dynamics can lead to suboptimal performance as the behavior of a system can alter during runtime. Rather, online learning techniques that are able to learn the actual policy from scratch and adapt themselves based on the current conditions appear to be more appropriate. To ensure optimal performance during runtime, the controller must observe the system dynamics and in case of large-scale mutations the control policy needs to be adjusted. By using fixed controllers this is infeasible. Instead, flexible procedures that identify modified performance and change towards the optimal behavior are required.

To deal with both, learning an optimal policy from scratch to avoid predefined physical models and updating the current policy as a result of changed conditions, we propose an online learning method based on the reinforcement learning principle. On the one hand, we train a model to represent the system dynamics. This model is learned based on collected experience and adapts automatically to the current behavior of the system by uprating newer observations. On the other hand, we keep up learning the value function according to the actual estimated behavior.

In the case of low dimensional state spaces where the transitions of each dimension can be trained, our approach leads to an efficient learning method based on reinforcement learning. The goal to be achieved by the controller is specified by virtual rewards given to the system when certain system states are reached. In our approach, the system learns online the dynamics by gathering transitions while the agent interacts with the environment. Based on these observations for each dimension an individual Gaussian process model is learned by adding training points iteratively. To obtain the policy given a predefined goal, the expected long-term reward of each state is learned via value iteration. Thereby, the uncertainties from the Gaussian processes representing the dynamics are taken into account. This function encodes for every state the next desirable state and allows for the extraction of the policy. Experimental results in two different scenarios demonstrate that our approach can quickly learn an optimal policy and, additionally, it can deal with changing system characteristics during runtime.

This paper is organized as follows. After discussing related work in the following section, we will briefly describe the fundamentals of reinforcement learning in the context of value iteration and introduce Gaussian process models. In section V we will present our approach to learn online a control policy. Section VI explains our sparsification methods and, finally, we will present experimental results obtained on an inverted pendulum and a real blimp robot.

II. Related Work

The problem of controlling a system has been studied intensively in the past. The approach that is most closely related to the one described in this paper is [2]. They also learn the system dynamics and the value function using separate Gaussian processes. The work described here, however, is an extension of their approach towards online learning based on sparsification methods. Due to the reduced time-complexity our approach can be applied to a wider range of applications.

Recently, in [3] a model-based reinforcement learning framework was proposed. As well, they learn both the dynamics based on previously collected experience and the policy by optimizing the value function. Gaussian process models are used as approximation and the sparsification

This work has partly been supported by the DFG within the Research Training Group 1103. Both authors are with the University of Freiburg, Department of Computer Science, D-79110 Freiburg, Germany

bases upon pseudo inputs [4]. In a related manner, [1], [5] presented dynamic programming including Gaussian processes (GPDP). They learn for individual state the Q-function to determine an overall value function. According to this value function and a classifier to distinguish between positive and negative actions the optimal policy is obtained. In comparison to these approaches, we learn the value function directly and introduce a sparsification criterion for online applications on real robots.

Additionally, Gaussian processes have been applied in several reinforcement learning tasks. An extension of temporal difference learning has been presented in [6], [7]. Their approach was successfully applied to the problem of learning complex manipulation policies [8]. Additionally, they presented a sparsification method for Gaussian processes where the point selection is related to our method. Finally, the sparsification is achieved by training a low rank covariance matrix according to a predefined maximum error. Compared to our approach, we project a new observation according to the current kernel onto the other support points which yields in a faster adaption by changing underlying functions.

III. Reinforcement Learning

Optimal control deals with the problem of finding a policy that allows an agent to act optimal towards a given goal. In the context of dynamic programming, the policy is typically defined by a function $\pi : S \rightarrow A$ which maps each state $\boldsymbol{s}$ to an action $\boldsymbol{a}$. In the reinforcement learning setting, a typical goal is to determine a value function that represents the expect long-term reward for each state. This value function for a continuous states space and a deterministic policy can be expressed by the Bellman equation [9]

$$V^{\pi}(\boldsymbol{s}) = \int_{s'} \mathcal{P}^{a}_{s,s'}[\mathcal{R}^{a}_{s,s'} + \gamma V^{\pi}(\boldsymbol{s}')]\,, \tag{1}$$

where $V^{\pi}(\boldsymbol{s})$ is the expected long-term reward of state $\boldsymbol{s}$ by following policy π. Furthermore, $\boldsymbol{a}$ is selected according to the policy, $\boldsymbol{a} = \pi(\boldsymbol{s})$, and $\mathcal{P}^{a}_{s,s'}$ is the transition probability of reaching state $\boldsymbol{s}'$ by executing action $\boldsymbol{a}$ in state $\boldsymbol{s}$. The term $\mathcal{R}^{a}_{s,s'}$ specifies the immediate reward to be received by taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$. Finally, $\gamma \in [0, 1]$ is a discount factor.

To solve (1) using dynamic programming, the dynamics of the system represented by the term $\mathcal{P}^{a}_{s,s'}$ needs to be known. Once the value function is determined, the optimal policy can easily be computed by

$$\pi(\boldsymbol{s}) \quad = \quad \arg\max_{a} \int_{s'} \mathcal{P}^{a}_{s,s'}[\mathcal{R}^{a}_{s,s'} + \gamma V(\boldsymbol{s}')]\,. \tag{2}$$

To find $\pi(\boldsymbol{s})$ one can use numerical methods taking advantage of the gradient w.r.t. the action [2]. For simplicity, we used a reasonable discretization of the action space.

IV. Gaussian Processes

A crucial choice for any learning task is the type of representation that is to be used for the core concepts. Most existing approaches operate on a discrete approximation over the state and action space. This, however, can lead to discretization errors or, when fine-grained grids are used, requires a huge amount of memory and a time-consuming exploration process. Therefore, function approximation techniques that directly operate on the continuous space such as neural networks [10], [11], kernel methods [12], or Gaussian processes [2], [7] have been proposed as powerful alternatives to the discrete approximations of continuous functions. From a regression perspective, these techniques seek to model the dependency

$$y_i = f(\boldsymbol{x_i}) + \epsilon_i \tag{3}$$

for the unknown function $f(\boldsymbol{x})$ and independent and identically, normally distributed noise terms ϵ_i, given a training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{D}$ of d-dimensional states $\boldsymbol{x}_i$ and target values y_i.

Gaussian processes can be seen as a generalization of weighted nearest neighbor regression [13] where the dependency of a function value on its local neighborhood is described using parameterized covariance functions. In this work, we apply the squared exponential

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \Lambda^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)\right), \tag{4}$$

where $\Lambda = diag(\ell_1, \ldots, \ell_d)$ is the length-scale, σ_f^2 the signal variance and $\boldsymbol{x}_i, \boldsymbol{x}_j$ are the inputs. Given a set of data samples $\mathcal{D}$ from the unknown function and the hyper-parameters $\boldsymbol{\theta} = (\sigma_f^2, \Lambda)$ of the covariance function, a Gaussian predictive distribution over function values for a new input location $\boldsymbol{x}^*$ can be determined by a normal distribution $\mathcal{N}(\mu^*, \sigma^*)$ with

$$\mu^* \quad = \quad \boldsymbol{k}(x, \boldsymbol{x}^*)^T \left(K + \sigma_n^2 I\right)^{-1} \boldsymbol{y} \tag{5}$$

$$\sigma^* \quad = \quad k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}(x, \boldsymbol{x}^*)^T \left(K + \sigma_n^2 I\right)^{-1} \boldsymbol{k}(x, \boldsymbol{x}^*)\,, \tag{6}$$

where $K \in \mathbb{R}^{D \times D}$ is the covariance matrix for the training points, σ_n^2 is the noise variance, $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{k}(x, \boldsymbol{x}^*) \in \mathbb{R}^D$ represents the covariances between the training points and the query point $\boldsymbol{x}^*$ with $k_i = k(\boldsymbol{x}_i, \boldsymbol{x}^*)$, and $\boldsymbol{y} = (y_1, \ldots, y_D)$ represents the values from the training set. For a detailed introduction to Gaussian process regression, we would like to refer the reader to [13]

In order to learn a Gaussian process one needs to acquire the training dataset $\mathcal{D}$ and to adapt the hyper-parameters $\boldsymbol{\theta}$. In this paper, the training points are gathered during the learning task as described in the next section and the hyper-parameters are optimized by maximizing the marginal data likelihood of the observed training data. The optimization is realized in a thread-based implementation. This yields a continuous training without having influences on the learning task.

V. Online Control

To learn the optimal policy in a potentially unknown environment, several approaches have been presented. Mostly, this is done by alternating the execution of an action and the integration of the resulting effects in the learning task. In the context of reinforcement learning, the agent is rewarded or penalized according to the action it performs. In general, the agent receives rewards for actions that are beneficial in certain states for achieving a long-term goal.

A common choice of solving the reinforcement learning problem is the Q-learning method and modifications of it [7], [14]. For each state-action pair, the expected long-term reward in reference to the subsequent states is estimated given a fixed policy. While the policy changes, a new Q-function needs to be learned. In situations where many states in the succeeding sequence need to be considered, several re-learning iterations are necessary. Actor-critic [9] is a common method to overcome this problem.

Our learning approach is an extension of value iteration method which bases on the Bellman Equation. As can be seen in (1), the expected long-term reward of state $\boldsymbol{s}$ depends only on the value of the following state $\boldsymbol{s}'$ given policy π. Therefore, the value $V(\boldsymbol{s})$ can easily be updated by determining the current best action according to (2) and reassigning the value $V(\boldsymbol{s})$ based on $V(\boldsymbol{s}')$. To achieve this, the transition function is required. Therefore, our method learns both, the system dynamics and the value function. The system dynamics are learned iteratively by gathering transitions from states-action pairs and approximating them by a Gaussian process. The value function is updated using policy evaluation according to the present dynamics model. In the following, we explain how the system dynamics can efficiently be learned and how the value function can be updated in alternating steps. The Gaussian processes representing the system dynamics and the value function are denoted as GP^d and GP^v respectively. In the same way, we define the hyper-parameters $\boldsymbol{\theta}^d$ and $\boldsymbol{\theta}^v$ as well as the training datasets $\mathcal{D}^d$ and $\mathcal{D}^v$.

A. System Dynamics

The system dynamics are gathered while the agent is interacting with the environment and approximated using a Gaussian process regression. Therefore, we learn for each state coordinate $s_c \in S_c, S = (S_1, \ldots, S_{|S|})$ a Gaussian process $GP_c^d : S \times A \rightarrow S_c$. At each time step t, while the agent executes an action $\boldsymbol{a}_t$, the transition is added to the individual datasets $\mathcal{D}_c^d$.

For learning tasks with huge numbers of state coordinates $|S|$, this is not manageable. In most cases, where online reinforcement learning is a suitable framework for real systems, this model could be learned efficiently.

B. Value Function

The value function is also approximated by a Gaussian process $GP^v : S \rightarrow V$. This function contains the expected long-term rewards for every state and, hence, describes how desirable it is to enter a certain state. To learn this function, (1) can be regarded as being an iterative update rule. This step is called policy evaluation and, in general, the values are reassigned until convergence. This assignment needs to be done for every state. In the following, we treat only one state.

Rasmussen and Kuss [2] proposed a method how to deal with uncertainties in the transition model $\mathcal{P}_{ss'}^a$ and how to take advantage of Gaussian processes in the evaluation step. As proposed, (1) can be split into two integrals

$$V^\pi(\boldsymbol{s}) = \int_{s'} \mathcal{P}_{s,s'}^a \mathcal{R}_{s,s'}^a + \gamma \int_{s'} \mathcal{P}_{s,s'}^a V^\pi(\boldsymbol{s}') \,. \tag{7}$$

The first integral can be computed directly for a polynomial or Gaussian reward function $\mathcal{R}_{ss'}^a$ where $\mathcal{P}_{s,s'}^a = \mathcal{N}(\boldsymbol{\mu}^d, \Sigma^d = diag(\sigma_1^2, \ldots, \sigma_{|S|}^2))$, in which mean and covariance are given coordinate-wise by GP_c^d. We identify the result of the first integral by I_R.

To solve the second integral, the prediction of the value at state $\boldsymbol{s}'$ is approximated by GP^v where the input itself is uncertain. For a detailed description of how to carry out such computations, we refer to [15]. In our case, the input state $\boldsymbol{s}'$ is again distributed according to GP^d by $\mathcal{N}(\boldsymbol{\mu}^d, \Sigma^d)$, which leads to the predictive distribution $p(V(\boldsymbol{s}') \mid \mathcal{D}^v, \boldsymbol{\theta}^v, \boldsymbol{\mu}^d, \Sigma^d)$. In case of Gaussian like covariance functions, this leads to

$$V^\pi(\boldsymbol{s}) \quad = \quad I_R + \gamma \boldsymbol{k}'(\boldsymbol{s}', x) K^{v-1} \boldsymbol{y} \,, \tag{8}$$

where x and $\boldsymbol{y}$ are given from $\mathcal{D}^v$ and k' indicates another covariance function with modified hyper-parameters $\boldsymbol{\theta}' = (\sigma_f^{2'}, \Lambda')$, with $\sigma_f^{2'} = |\Lambda^{v-1}\Sigma^d + \boldsymbol{I}|^{-1/2}\sigma_f^{v2}$, and $\Lambda' = \Lambda^v + \Sigma^d$. Intuitively, this is the prediction of $\boldsymbol{s}'$ with changed covariances between the point $\boldsymbol{s}'$ and the training set. In the case of online learning, training the system dynamics GP^d and updating the values with policy evaluation are performed in an alternating fashion.

Please note the training points $\mathcal{D}^v$ can be added iteratively during the learning task or equally loaded at the beginning. The latter is very efficient if an operating range in the state space can be specified. The maximum number of manageable states can be loaded to guarantee in-time processing. Also a combination of both the initialization with a fixed set of points and the iterative adding of new points is conceivable.

An off-line method of this learning approach has been proposed in [2]. The system dynamics are trained given some previously gathered observations. Afterwards, the policy is evaluated until convergence. Compared to our approach, we learn the system dynamics and the policy in alternating steps and sparsification methods can easily be applied. Furthermore, in the case of online control, it is possible to identify changing conditions. Affected transitions in the system dynamics can be adapted and a new value function can be evaluated.

VI. Sparsification

In order to apply the approach presented above to real learning tasks, we seek to reduce the dataset $\mathcal{D}$. This leads to a smaller time-complexity conditional upon the inverse covariance matrix K. A common approach for sparsification of Gaussian process models is to use a subset of the datapoints only and to approximate the influence of the excluded ones. In the case of adding points iteratively, the informative vector machine [16], for instance, selects points based on an entropy criterion. This turns out to favor sampling in high-variance areas. Since a Gaussian process' predictive variance depends on the local data density only and not on the target values, this basically achieves an uniform distribution of

input points. Note that [7] basically also use this approach to limit their dictionary set.

As there is a direct relationship between the local data density, the kernel parameters, and a Gaussian process' predictive variance, we can approximate the variance-based sparsification criterion by a criterion based on distances in input space. This can be quickly evaluated, $\Delta = \min_i \boldsymbol{x}_i^T \boldsymbol{x}'$, where $\boldsymbol{x}'$ is the new point and $i \in \{1, \ldots, D\}$ passes through the dataset of support points. If $\Delta > \nu$ as a defined threshold, the point is added. Equally, to factor the individual dimensions into the criterion the kernel parameters $k(.,.)$ can be used. This will improve the selection as high correlated dimensions need less points to be approximated. Additionally, the control-space and the available computational resources can be used to limit the manageable number of points. Note that our criterion avoids the computation of predictive variances, while resulting in a similar, close to uniform distribution of support points. To model highly nonlinear functions uniform distributions are suboptimal and other covariance functions are preferable. However, using our approach control tasks, as presented in the experiments, can be performed online as the sparsification criterion can be determined very efficient.

Furthermore, in the case of control the optimal policy is bounded to the system dynamics. If the system dynamics change, the policy needs to be updated. Therefore, in contrast to the perspective of data fitting, we seek to approximate the unknown function with higher importance of recent observations. This, however, implies that it is needless to store previous knowledge such as in most other sparsification methods. As well, if the kernel parameters are trained simultaneously, recursive update formulas, which easily deal with new observations, could not be adhered.

To be adaptable for changing underlying functions, if a new observation $(\boldsymbol{x}', y')$ is not added to the dataset, we map the information according to the current kernel parameters onto all other support points. The conditional distribution $\mathcal{D}_{t+1} \mid \mathcal{D}_t, \boldsymbol{x}', y'$ for the target values is given by

$$\boldsymbol{y}_{\mathcal{D}_{t+1}} = \boldsymbol{y}_{\mathcal{D}_t} + \boldsymbol{k}(\boldsymbol{x}', x_{\mathcal{D}_t})^T \sigma^{-2} (y' - \mu_{x'}) \,, \tag{9}$$

where $\boldsymbol{k}(\boldsymbol{x}', x_{\mathcal{D}})^T \in \mathbb{R}^D$ represents the covariance between the new point $\boldsymbol{x}'$ and the training points, σ^2 the noise of the observed target value y', and $\mu_{x'}$ the prediction of x' according to the Gaussian process model. Please note only the target values of the dataset are modified. The input states and the total number of points remain unchanged.

Since the targets of the value function are evaluated according to (8), an update according to (9) is omitted. Consequently, only the approximation of the system dynamics is updated according to (9). The resulting procedure for adding value states iteratively is outlined in Algorithm 1.

Algorithm 1 Online value iteration.

for $t = 0, 1, \ldots$ **do**
  Determine action $\boldsymbol{a}_t$ according to (2) and observe $\boldsymbol{s}_{t+1}$
  Compute distances Δ^d and Δ^v
  if $\Delta^d > \nu^d$ **then**
    Add for each state coordinate the transition to the corresponding GP^d
  else
    Map new observation onto $\mathcal{D}^d$ according to (9)
  end if
  Optimize hyper-parameters $\boldsymbol{\theta}^d$
  if $\Delta^v > \nu^v$ **then**
    Add state $\boldsymbol{s}_t$ to GP^v and initialize the target with

$$V(\boldsymbol{s_t}) = \mathcal{R}^{a_t}_{s_t, s_{t+1}} + \gamma V(\boldsymbol{s}_{t+1})$$

  end if
  Update targets of GP^v according to (8)
  Optimize hyper-parameters $\boldsymbol{\theta}^v$
end for

VII. Experiments

The approach described above has been implemented and tested in two different scenarios. The goal of the experiments is to demonstrate that our approach is able to learn online to control a system without any prior information about the behavior within a short time-frame. In the first experiment, we apply our learning approach to the task of controlling an inverted pendulum. Afterwards, we analyze our method in stabilizing a blimp at a given goal altitude. Finally, we show that our approach can deal with changing conditions and is able to quickly adapt the policy. During all these experiments we applied the ϵ-greedy policy with $\epsilon = 0.1$.

A. Inverted Pendulum

In the first experiment, we consider the under-powered pendulum. The goal is to swing up an inverted pendulum from the start state $[\varphi, \dot{\varphi}]^T = [-\pi, 0]^T$ to the goal state $[0, 0]^T$ and balance it in this position. Angle and angular velocity are denoted by φ and $\dot{\varphi}$ respectively. The challenge lies in recognizing to first swing in one direction and afterwards accelerate back to the start state with enough momentum to go towards the goal state. This means, during learning the agent must incur additional negative immediate rewards.

To perform this experiment, we simulate the system dynamics following ODE. According to [1], the acceleration is given by

$$\ddot{\varphi}(t) = \frac{-\mu \dot{\varphi}(t) + mgl \sin(\varphi(t)) + a(t)}{ml^2} \,, \tag{10}$$

where $l = 1\,\mathrm{m}$ is the length, $m = 1\,\mathrm{kg}$ is the mass, $\mu = 0.05\,\frac{\mathrm{kg\,m^2}}{\mathrm{s}}$ is the friction, and $g = 9.81\,\frac{\mathrm{m}}{\mathrm{s^2}}$ is the gravitational constant. Adapted from (10) and by discretizing the time, we obtain

$$\boldsymbol{s}_{t+1} = \begin{bmatrix} \varphi_{t+1} \\ \dot{\varphi}_{t+1} \end{bmatrix} = \begin{bmatrix} \varphi_t + \Delta t \dot{\varphi}_t + \frac{\Delta t^2}{2} \ddot{\varphi}_t \\ \dot{\varphi}_t + \Delta t \ddot{\varphi}_t \end{bmatrix} . \tag{11}$$

As sampling rate we use 10 Hz and added small amounts of white noise ($\sigma = 0.01$). An episode considers 5 seconds and we set $\mathcal{R}^a_{ss'} = -s^2$ as immediate reward function.

To evaluate our approach, after each episode an evaluation run were carried out. Therefore, in this run the pendulum

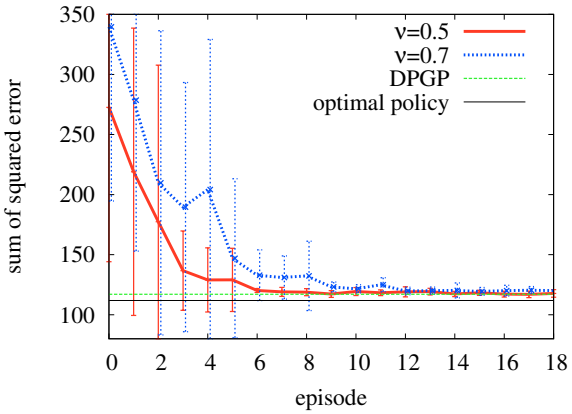

Fig. 1. Learning rate of our online value iteration approach applied to the inverted pendulum. As evaluation criterion, we calculate the sum of the squared distances to the goal position in evaluation runs.

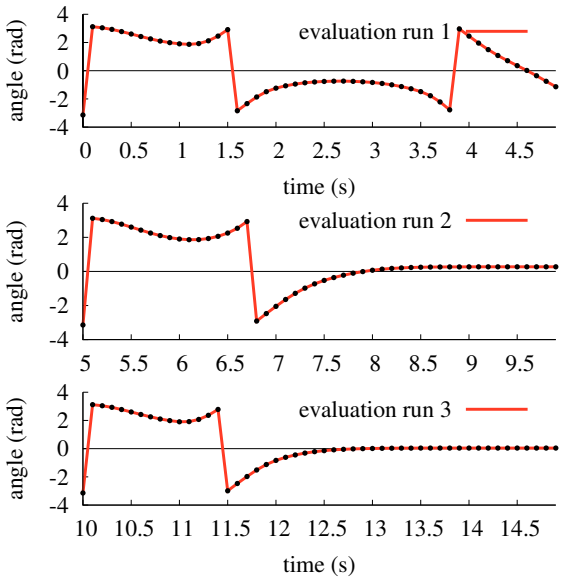

Fig. 2. Evolution of the angle φ during the learning process for the inverted pendulum.

were actuated by the greedy actions according to the so far learned policy. The sum of squared distances of the state angle φ provides as a benchmark criterion. The learning rate averaged over multiple runs and varying sparsification thresholds ν are shown in Fig. 1. The optimal policy generates an error of 111.9 and was determined using dynamic programming based on the ODE. As well, the error obtained by using a policy learned with GPDP [1] is shown. To achieve this policy 400 points were randomly sampled in the state space. In contrast to our approach, we achieve the same behavior after the fifth episode with e.g. 110 points in $\mathcal{D}^v$, 140 points in $\mathcal{D}^d$, and $\nu = 0.5$. Fig. 2 illustrates state φ of the evaluation runs. As can be seen, our approach is able to balance the pendulum after to second learning episodes.

B. Learning Online with a Real Robot

The second experiment is designed to demonstrate that our approach can be applied online with a real robot. Here, the goal is to stabilize the height of a blimp by learning the policy from scratch without knowing the specific system dynamics or any parameters of the environment. To carry out this experiment, we used a miniature indoor blimp with a length of 1.8 m and a diameter of 0.9 m [17]. The blimp is steered by three propellers whereof we only actuate the two main propellers. They were oriented upwards to control the vertical motion of the airship. A downward-facing ultrasound sensor was used to measure the height of the system. The measurements were integrated on-board by a Kalman filter which sequentially estimates the height and the vertical velocity. This data were send via wireless to a standard laptop computer (1.6 GHz) that performs the learning task.

We define the state space S by $s_t = (d_t, v_t)$, where $d_t = h^* - h_t$ represents the distance to the goal altitude h^* and v_t the vertical velocity of the blimp. The action space A is a discretized interval $a_t \in [-1, 1]$ with a resolution of 0.05. To perform this experiment, we initialize the dataset $\mathcal{D}^v$ at the beginning with 100 randomly distributed points. The dynamics were learned with a fixed time interval between the proceeding measurements of $\Delta t = 0.5$ s. As sparsification criterion for GP^d we set $\nu = 0.04$. Furthermore, as immediate reward function we used the distance to the target height $\mathcal{R}^a_{ss'} = -(d_t)^2$.

To perform this experiment, we ran the blimp in a factory building with a vertical exploration space of 5 m. To achieve a learning system running online on a standard laptop computer and to avoid the blimp having to wait for the learning system, we realized a thread-based implementation which performs the optimization of the Gaussian processes in parallel to the learning task. The upper graph of Fig. 3 illustrates a typical evolution of the height during a learning task. As can be seen, at the beginning the blimp is exploiting the states and with proceeding learning time it stabilize at the goal altitude of 2 m. Furthermore, the corresponding error to a base-line policy is shown in the lower graph. The base-line policy was computed using dynamic programming with a transition function based on the dynamics of the system. Finally, in this run 69 points were used as training set $\mathcal{D}^d$ and one update step of all 100 value points needs about 0.25 s.

C. Identification of Changed Behavior

The final experiment is designed to illustrate the benefit of our learning approach in the context of controlling a system. If the characteristics of a system alters during operation, this needs to be identified and the controller should be adapted to keep up optimality. To demonstrate the behavior of our learning process in such situations, we manually modified the environmental conditions. Concretely, after 40 seconds we simulate loosing buoyancy by increasing the mass of the blimp. As new observations are continuously integrated according to (9), the dynamics are successive adapted to the current behavior. To perform this experiment, we started the blimp with a previously learned policy in simulation as shown in Fig. 4. The upper graph shows the altitude progress

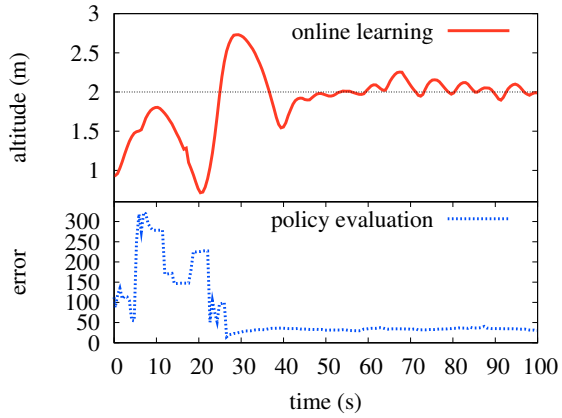

Fig. 3. Altitude progress of the real blimp while it learns online to stabilize at 2 m. In the lower graph the corresponding error to the base-line policy is shown.

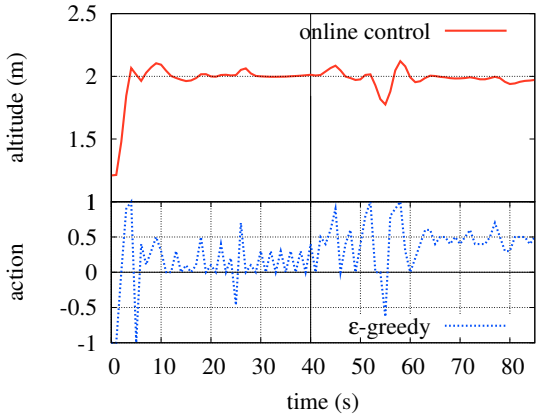

Fig. 4. Altitude progress to stabilize at 2 m using a previous learned policy. At time-step 40 m, we manually changed the environmental conditions by increasing the mass of the blimp to simulate loss of buoyancy. After a short period of re-learning, the blimp stabilizes again by increasing the throttle control.

and the lower the selected actions. At the beginning the system stabilize at the goal height of 2 m. As the behavior of the blimp changes suddenly at time-step 40 s, the policy needs to be updated. As the dynamics are adjusted to the new characteristics the value function is as well adapted towards the current conditions. As can be seen, the system re-learns quite fast by readjusting the value function which results in an increased throttle control.

If standard Monte Carlo methods are performed endless and, in case of Q-learning, if all state-action pairs are visited, they will also converge [9]. For this reason, if the behavior of a system changes, in a long-term run standard learning methods will always improve towards the optimal policy. However, the presented approach is beneficial based on the fact that the dynamics are separately modeled and can quickly adapt to changing conditions.

VIII. Conclusions

In this paper, we presented a novel approach to online learning control policies of physical systems. Our method connects both, an reinforcement learning method using Gaussian processes to approximate the core functions as well as online sparsification methods to reduce the time complexity. Our approach calculates in alternating processes the effects of actions and determines the expected long-term rewards of the individual states. Additionally, the learned system dynamics can be quickly adapted to changed conditions and, therefore, an optimal behavior can be achieved even if the characteristics alter during runtime. In practical experiments, we demonstrated that our method can be applied to typical control tasks such as the inverted pendulum and to a robotic blimp system which is highly sensitive to outer influences and hard to control.

References

[1] M. P. Deisenroth, J. Peters, and C. E. Rasmussen, "Approximate dynamic programming with gaussian processes," in *Proc. of 2008 American Control Conference (ACC)*, 2008.

[2] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Proc. of the Advances in Neural Information Proc. Systems 16 (NIPS)*. MIT Press, 2004.

[3] C. E. Rasmussen and M. P. Deisenroth, *Recent Advances in Reinforcement Learning*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2008, vol. 5323, lecture notes in artificial intelligence Probabilistic Inference for Fast Learning in Control.

[4] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Proc. of the Advances in Neural Information Proc. Systems 18*. MIT Press, 2006.

[5] M. P. Deisenroth, C. E. Rasmussen, and J. Peters, "Model-Based Reinforcement Learning with Continuous States and Actions," in *Proc. of the 16th Europ. Symp. on Artificial Neural Networks (ESANN)*, 2008.

[6] Y. Engel, S. Mannor, and R. Meir, "Bayes meets Bellman: The Gaussian Process approach to temporal difference learning," in *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2003.

[7] ——, "Reinforcement learning with gaussian processes," in *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2005.

[8] Y. Engel, P. Szabo, and D. Volkinshtein, "Learning to control an octopus arm with gaussian process temporal difference methods," in *Advances in Neural Information Processing Systems (NIPS)*, 2005.

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

[10] R. Coulom, "Reinforcement learning using neural networks, with applications to motor control," Ph.D. dissertation, Institut National Polytechnique de Grenoble, 2002.

[11] M. Riedmiller, "Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method," in *Proc. of the Europ. Conf. on Machine Learning (ECML)*, 2005.

[12] N. Jong and P. Stone, "Kernel-based models for reinforcement learning in continuous state spaces," in *ICML workshop on Kernel Machines and Reinforcement Learning*, June 2006.

[13] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[14] A. Rottmann, C. Plagemann, P. Hilgers, and W. Burgard, "Autonomous blimp control using model-free reinforcement learning in a continuous state and action space," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[15] A. Girard, C. E. Rasmussen, J. Quiñonero-Candela, and R. Murray-Smith, "Multiple-step ahead prediction for non linear dynamic systems - a Gaussian process treatment wih propagation of the uncertainty," in *Proc. of the Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

[16] N. Lawrence, M. Seeger, and R. Herbrich, "Fast sparse gaussian process methods: The informative vector machine," in *Advances in Neural Information Processing Systems 15 (NIPS)*. MIT Press, 2003.

[17] A. Rottmann, M. Sippel, T. Zitterell, W. Burgard, L. Reindl, and C. Scholl, "Towards an experimental autonomous blimp platform," in *Proc. of the 3rd Europ. Conf. on Mobile Robots (ECMR)*, 2007.