Approximate Real-Time Optimal Control Based on Sparse Gaussian Process Models

Joschka Boedecker*, Jost Tobias Springenberg*, Jan Wülfing*, Martin Riedmiller
Machine Learning Lab, Department of Computer Science
University of Freiburg, 79110 Freiburg, Germany
Email: {jboedeck,springj,wuelfj,riedmiller}@informatik.uni-freiburg.de
*authors contributed equally

***Abstract*—In this paper we present a fully automated approach to (approximate) optimal control of non-linear systems. Our algorithm jointly learns a non-parametric model of the system dynamics – based on Gaussian Process Regression (GPR) – and performs receding horizon control using an adapted iterative LQR formulation. This results in an extremely data-efficient learning algorithm that can operate under real-time constraints. When combined with an exploration strategy based on GPR variance, our algorithm successfully learns to control two benchmark problems in simulation (two-link manipulator, cart-pole) as well as to swing-up and balance a real cart-pole system. For all considered problems learning from scratch, that is without prior knowledge provided by an expert, succeeds in less than 10 episodes of interaction with the system.**

I. Introduction

Reinforcement Learning (RL) from scratch has seen some impressive applications to real-world problems in recent years (e.g. [1], [2], [3], see [4] for a recent survey of RL in robotics). However, most RL methods either rely on global value function approximation which requires complex regression models and large amounts of samples for high dimensional problems, or directly learn a policy using model-free policy search methods. This circumvents the value function approximation problem, but is typically data inefficient, requiring hundreds of episodes before reaching a solution.

Model-based RL methods, on the other hand, hold the promise of achieving both data efficiency as well as control of complex dynamical systems. Among them, local trajectory optimization methods [5], [6] based on the principles of (stochastic) optimal control are an attractive alternative to global RL methods since they are not directly affected by the curse of dimensionality. Similarly, model-based policy search methods can search the policy space offline using a system model – hence minimizing the interactions with the real system – and discover successful policies on complex dynamical systems in as few as 10 episodes [7], [8]. These methods usually assume the existence of an exact dynamics model of the system to be controlled. Such models rely on – and are restricted by – the insight of the controller designer. Furthermore, model specification is a laborious process, and difficulties arise if information on parts of the system is not available, or subject to change.

A promising approach then is to use machine learning techniques to identify a system model automatically from data generated by interacting with the system. In addition to being applicable to arbitrary system dynamics, learning methods can be adaptive to changes in the system dynamics over time and even account for effects that are hidden from a human observer (such as material wear). Learned dynamics models have been used successfully in a substantial number of cases (see [9] for a relevant survey). However, any learned model will likely be biased since only a limited number of data points can be collected on most real systems. This leads to inaccuracies due to undersampling of the state-space. Trajectory optimization methods, are particularly vulnerable to inaccuracies in a system model, since they rely on derivative information in addition to forward predictions. As pointed out in [10], this can lead to blow-ups during forward integration. Furthermore, for real-time control and the small control intervals it requires, the execution time for a given model quickly becomes a limiting factor. These problems taken together have hampered the successful application of model-based trajectory optimization to learning from scratch (i.e. without including large amounts of prior knowledge about the system).

In this work we present a fully automated approach for (approximate) real-time optimal control that is applicable to real-world systems. We propose an algorithm, which we dub *Approximate iterative LQR with Gaussian Processes* (AGP-iLQR), that is based on trajectory optimization with iLQR (iterative Linear Quadratic Regulator) [5]. It uses a non-parametric dynamics model based on Gaussian Processes (GPs), estimated efficiently from interaction data. We show how GPs can effectively be used in the iLQR framework, and how a GP model can be learned alongside the trajectory optimization using a sparse approximation. To fight model-bias, we control the system in a receding horizon fashion and use an exploration scheme based on the "optimism in the face of uncertainty" principle [11] which uses the GP uncertainty to guide trajectory optimization. To facilitate real-time control we use one-step-ahead predictions of the optimal control law (which can be computed concurrently to the control loop).

We demonstrate the efficacy of our approach on two simulated tasks and on a real cart-pole system, which, to the best of our knowledge, has not been solved from scratch with optimal control approaches based on learned models before.

II. Foundations

Before we describe our approach we formally introduce the underlying control problem as well as the two key components that constitute the basis of our algorithm: (1) iLQR, a trajectory optimization algorithm from the differential dynamic programming family; (2) GPR for estimating system dynamics.

978-1-4799-4552-8/14/$31.00 ©2014 IEEE

A. Problem Definition

We consider the control of dynamical systems of the form

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w},\ \mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_w), \tag{1}$$

where states $\mathbf{x} \in \mathbb{R}^D$, controls $\mathbf{u} \in \mathbb{R}^F$ as well as control noise $\mathbf{w}$ are continuous-valued, and f is an arbitrary dynamics function. The goal of stochastic optimal control then is to find (locally) optimal controls $\mathbf{u}^*_{1:T}$ for the limited horizon T, which realize a corresponding optimal trajectory, $\mathbf{x}^*_{1:T}$ by maximizing the expected reward of the trajectory

$$J(\mathbf{x}_{1:T}, \mathbf{u}_{1:T}) = \mathbb{E}_f[\sum_{t=1}^{T} r(\mathbf{x}_t, \mathbf{u}_t)], \tag{2}$$

where $r(\mathbf{x}_t, \mathbf{u}_t) \in \mathbb{R}$ is the immediate reward function and $E_f[\cdot] = E_{\mathbf{x}_{t+1}=f(\mathbf{x}_t,\mathbf{u}_t)+\mathbf{w}}[\cdot]$ is the expectation under known dynamics f. The time-dependent value function V_t, corresponding to this optimization problem, can then be written as

$$V_t(\mathbf{x}_t) = \max_{\mathbf{u}_{t:T}} J(\mathbf{x}_{t:T}, \mathbf{u}_{t:T}). \tag{3}$$

B. Trajectory Optimization with iLQR

Iterative LQR [5] is a powerful trajectory optimization method from the family of differential dynamic programming (DDP) [12] algorithms. The core idea behind DDP is to find a locally optimal trajectory (and optimal local controls), for *known* system dynamics f and rewards r using an iterative procedure. It proceeds by, first, linearizing f forward in time around the current best trajectory for which a locally optimal control law can then be computed backwards in time. This gives rise to an improved trajectory, which can again be optimized using the described forward and backward calculations. These steps are then repeated until convergence to the locally optimal trajectory [5].

Formally, following the derivation from [13], let $\bar{\mathbf{x}}_{1:T} = [\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_T]$ denote a reference trajectory given as a sequence of states and $\bar{\mathbf{u}}_{1:T} = [\bar{\mathbf{u}}_1, \dots, \bar{\mathbf{u}}_T]$ denote the corresponding actions. We obtain an approximate model of the system dynamics at time t w.r.t. the reference trajectory by linearization as

$$\mathbf{x}_{t+1} - \bar{\mathbf{x}}_{t+1} \approx f_{\mathbf{x}t}(\mathbf{x}_t - \bar{\mathbf{x}}_t) + f_{\mathbf{u}t}(\mathbf{u}_t - \bar{\mathbf{u}}_t), \tag{4}$$

and a quadratic model of the rewards via a second-order Taylor expansion

$$\begin{aligned} r(\mathbf{x}_t, \mathbf{u}_t) \approx\ & r(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t) + \delta^T_{\mathbf{x}_t} r_{\mathbf{x}t} + \delta^T_{\mathbf{u}_t} r_{\mathbf{u}t} \\ & + \frac{1}{2}\delta^T_{\mathbf{x}_t} r_{\mathbf{x}\mathbf{x}t}\delta_{\mathbf{x}_t} + \frac{1}{2}\delta^T_{\mathbf{u}_t} r_{\mathbf{u}\mathbf{u}t}\delta_{\mathbf{u}_t} \\ & + \delta^T_{\mathbf{x}_t} r_{\mathbf{x}\mathbf{u}t}\delta_{\mathbf{u}_t}, \end{aligned} \tag{5}$$

where $\delta_{\mathbf{x}_t} = (\mathbf{x}_t - \bar{\mathbf{x}}_t)$, $\delta_{\mathbf{u}_t} = (\mathbf{u}_t - \bar{\mathbf{u}}_t)$ are deviations from the reference trajectory and the terms with subscripts denote Jacobian and Hessian matrices of their respective functions.

In combination this *linear-quadratic* formulation now can be used to compute a quadratic approximation to the *time-dependent* value function V_t (and its corresponding state-action value function Q_t) by, again, taking a Taylor expansion along the trajectory. The individual Jacobian and Hessian terms (which can be computed recursively *backwards* in time starting at time-point t) are:

$$\begin{aligned} \mathbf{Q}_{\mathbf{x}\mathbf{x}t} &= \mathbf{r}_{\mathbf{x}\mathbf{x}t} + f^T_{\mathbf{x}t}\mathbf{V}_{\mathbf{x}\mathbf{x}t+1}f_{\mathbf{x}t} & \mathbf{Q}_{\mathbf{x}t} &= \mathbf{r}_{\mathbf{x}t} + f^T_{\mathbf{x}t}\mathbf{V}_{\mathbf{x}t+1} \\ \mathbf{Q}_{\mathbf{u}\mathbf{u}t} &= \mathbf{r}_{\mathbf{u}\mathbf{u}t} + f^T_{\mathbf{u}t}\mathbf{V}_{\mathbf{x}\mathbf{x}t+1}f_{\mathbf{u}t} & \mathbf{Q}_{\mathbf{u}t} &= \mathbf{r}_{\mathbf{u}t} + f^T_{\mathbf{u}t}\mathbf{V}_{\mathbf{x}t+1} \\ \mathbf{Q}_{\mathbf{u}\mathbf{x}t} &= \mathbf{r}_{\mathbf{u}\mathbf{x}t} + f^T_{\mathbf{u}t}\mathbf{V}_{\mathbf{x}\mathbf{x}t+1}f_{\mathbf{x}t}, \end{aligned} \tag{6}$$

and corresponding value function terms:

$$\begin{aligned} \mathbf{V}_{\mathbf{x}t} &= \mathbf{Q}_{\mathbf{x}t} - \mathbf{Q}^T_{\mathbf{u}\mathbf{x}t}\mathbf{Q}^{-1}_{\mathbf{u}\mathbf{u}t}\mathbf{Q}_{\mathbf{u}t} \\ \mathbf{V}_{\mathbf{x}\mathbf{x}t} &= \mathbf{Q}_{\mathbf{x}\mathbf{x}t} - \mathbf{Q}^T_{\mathbf{u}\mathbf{x}t}\mathbf{Q}^{-1}_{\mathbf{u}\mathbf{u}t}\mathbf{Q}_{\mathbf{u}\mathbf{x}t}. \end{aligned} \tag{7}$$

The reference controls $\bar{\mathbf{u}}_{1:T}$ can then be improved to $\bar{\mathbf{u}}^{\text{next}}_{1:T}$ using a (deterministic) policy $\bar{\pi}$ given as the locally linear feedback controller:

$$\begin{aligned} \bar{\mathbf{u}}^{\text{next}}_t &= \bar{\pi}(\mathbf{x}_t) = \bar{\mathbf{u}}_t + \mathbf{g}_t + \mathbf{G}_t(\mathbf{x}_t - \bar{\mathbf{x}}_t), \\ \text{with gains } \mathbf{g}_t &= -\mathbf{Q}^{-1}_{\mathbf{u}\mathbf{u}t}\mathbf{Q}_{\mathbf{u}t}, \\ \mathbf{G}_t &= -\mathbf{Q}^{-1}_{\mathbf{u}\mathbf{u}t}\mathbf{Q}_{\mathbf{u}\mathbf{x}t}. \end{aligned} \tag{8}$$

Finally, $\bar{\mathbf{u}}^{\text{next}}_{1:T}$ can be applied *forward* in time (using dynamics f) to retrieve a new reference trajectory $\bar{\mathbf{x}}^{\text{next}}_{1:T}$.

In order to arrive at $\mathbf{u}^*_{1:T}$ and $\mathbf{x}^*_{1:T}$ this procedure is iterated until convergence. Notice that this will result in a locally optimal open loop control sequence $\mathbf{u}^*_{1:T}$ which – when applied to a real system – might not realize $\mathbf{x}^*_{1:T}$ due to inaccuracies in the assumed system model.

To achieve robust control we therefore adopt a model predictive control approach using the receding horizon scheme introduced in [14]. That is, in a given state $\mathbf{x}_t$, instead of using the open loop controls $\mathbf{u}^*_{1:T}$, we only execute $\mathbf{u}^*_1$, observe the real next state $\mathbf{x}_{t+1}$ and warm-start another iLQR optimization using the time-shifted controls $\overleftarrow{\mathbf{u}}^*_{1:T} = [\mathbf{u}^*_2, \dots, \mathbf{u}^*_T, \mathbf{u}^*_T]$ and corresponding reference trajectory $\overleftarrow{\mathbf{x}}^*_{1:T} = [\mathbf{x}_{t+1}, \mathbf{x}^*_3, \dots, \mathbf{x}^*_T, f(\mathbf{x}^*_T, \overleftarrow{\mathbf{u}}^*_T)]$. If we assume that f is sufficiently accurate then the deviation $\|\mathbf{x}_{t+1} - \mathbf{x}^*_2\|_2$ will be small and the warm-started optimization will converge in few iterations (1-2 in practice) since $\overleftarrow{\mathbf{u}}^*_{1:T}$ is already close to the optimal controls.

C. Learning a Model with Gaussian Process Regression

Gaussian Processes (GPs) are a popular choice for model learning [15], [16], [2], which are extremely data efficient (utilizing available training data in a Bayes-optimal way) and hence are well suited for our needs for learning a dynamics model. We therefore adopt a Gaussian Process Regression (GPR) model for f following the formulation from [2].

1) Learning the dynamics function of the system: Let f denote the dynamics function as defined in Eq. (1) and $\tilde{\mathbf{x}}_{1\dots N} = [\tilde{\mathbf{x}}_1 \dots \tilde{\mathbf{x}}_N]$ denote the given training data consisting of an observed sequence of N states, where system states $\mathbf{x}$ and controls $\mathbf{u}$ are combined to form the vector of extended states $\tilde{\mathbf{x}} = [\mathbf{x}, \mathbf{u}]^T$. In order to specify a GP, we need to define a mean function $m(\cdot)$ and a covariance function (also called kernel) $k(\cdot, \cdot)$. Different functions can be used depending on the problem at hand. Here, we use $m \equiv 0$ and the squared exponential kernel with automatic relevance determination defined as

$$k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_p \frac{(\bar{x}_{i,p} - \bar{x}_{j,p})^2}{l_p^2}\right) + \delta_{ij}\sigma_w^2. \tag{9}$$

The characteristic length-scales $l_1, \dots l_{D+F}$ are hyperparameters that need to be optimized (see Section III-D). Further hyperparameters are σ_f^2 and $\sigma_{\mathbf{w}}^2$ for the variance of the latent dynamics function f and the system noise $\mathbf{w}$. Rather than training the GP model $\mathcal{M}$ to predict next states $\tilde{\mathbf{x}}_{t+1}$ from $\tilde{\mathbf{x}}_t$ we train it to predict changes in the system state given applied controls, i.e. for a given $\tilde{\mathbf{x}}_t$, the training target is $\boldsymbol{\Delta}_t = \mathbf{x}_{t+1} - \mathbf{x}_t$. Predicting state differences instead of next states was previously found to be advantageous [2]. Importantly, it makes efficient use of the GPs zero mean prior, resulting in an implicit prior mean function over states $m(\mathbf{x}) = \mathbf{x}$. That is, for unknown inputs, the GP will assume that the system state does not change. For the high-frequency control loops we are aiming for, this is a reasonable assumption, which can also help generalization to novel states and thus eliminate spurious predictions.

The posterior $p_{\mathcal{M}}(\boldsymbol{\Delta}_t|\tilde{\mathbf{x}}_t)$ of the GP model $\mathcal{M}$ is a Gaussian random variable with mean

$$\mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t] = m_{\mathcal{M}}(\tilde{\mathbf{x}}) = \mathbf{k}_*^T(\mathbf{K} + \sigma_{\mathbf{w}}^2\mathbf{I})^{-1}\mathbf{y} \tag{10}$$

and variance

$$\mathbb{V}_{\mathcal{M}}[\boldsymbol{\Delta}_t] = k_{**} - \mathbf{k}_*^T(\mathbf{K} + \sigma_{\mathbf{w}}^2\mathbf{I})^{-1}\mathbf{k}_*, \tag{11}$$

with $\mathbf{k}_* = [k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}_1), \dots, k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}_N)]^T$, $k_{**} = k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}})$, and $\mathbf{K} \in \mathbb{R}^{N\times N}$ is the covariance matrix whose entries are defined as $K_{ij} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$.

To use the GP Model $\mathcal{M}$ for estimating the dynamics f we can now observe that for any input $\tilde{\mathbf{x}}_t$, the predictive posterior for the successor state $\mathbf{x}_{t+1}$ is also a Gaussian distributed random variable, i.e.

$$p_{\mathcal{M}}(\mathbf{x}_{t+1}|\,\tilde{\mathbf{x}}_t) = \mathcal{N}(\mathbf{x}_{t+1} \mid \mathbf{x}_t + \mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t], \mathbb{V}_{\mathcal{M}}[\boldsymbol{\Delta}_t]). \tag{12}$$

Therefore f from Eq. (1) can be approximated as:

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx \mathbb{E}_{\mathcal{M}}[\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t] = \mathbf{x}_t + \mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t]. \tag{13}$$

It should be noted that in general, our targets are multivariate and we train one GP per target dimension. These individual GPs are conditionally independent for given test inputs.

III. Real-Time Gaussian Process based iLQR

Building on the foundations from Section II we will now describe our proposed method for receding-horizon closed loop control.

A. Gaussian Process based iLQR

In order to use a GP model within iLQR we need to be able to efficiently compute the individual Q-terms from Section II-B above. In turn, this requires a closed-form approximation for the derivatives f_{*t} of the dynamics function. Under the GP approximation this reduces to computing derivatives of the posterior of the dynamics $p_{\mathcal{M}}(\mathbf{x}_{t+1}|\tilde{\mathbf{x}}_t)$ as estimated by the GP. Since taking a derivative is a linear operation, the derivative of a GP is another, dependent, GP [17]. This derived GP can be used to make predictions for f_{*t}. Note that these are predictions of spatial derivatives w.r.t. $\mathbf{x}_t$ and $\mathbf{u}_t$ which are therefore *not* equivalent to the differences $\boldsymbol{\Delta}_t$ used as targets for the GP model.

1) Generating derivative information: Formally, let $\mathcal{M}$ be the GP model for predicting state differences $\boldsymbol{\Delta}$ with mean $\mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t]$ and variance $\mathbb{V}_{\mathcal{M}}[\boldsymbol{\Delta}_t]$. The derivative GP Model denoted as $\mathcal{M}'$ computes predictions for $\boldsymbol{\Delta}'_t$ which are Gaussian distributed with mean:

$$\mathbb{E}_{\mathcal{M}'}[\boldsymbol{\Delta}'_t] = \frac{\partial \mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t]}{\partial \tilde{\mathbf{x}}_t} = \mathbf{J}^T(\mathbf{K} + \sigma_{\mathbf{w}}^2\mathbf{I})^{-1}\mathbf{y}, \tag{14}$$

where $\mathbf{J}$ is the Jacobian matrix of partial first derivatives of the kernel function w.r.t. the components of vector $\tilde{\mathbf{x}}_t$. Hence the ith column of $\mathbf{J}$ is the derivative $\frac{\partial k(\tilde{\mathbf{x}}_t, \tilde{\mathbf{x}}_i)}{\partial \tilde{\mathbf{x}}_t}$, or more precisely

$$J_{*,i} = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_p \frac{(\bar{x}_{t,p} - \bar{x}_{i,p})^2}{l_p^2}\right)(\tilde{\mathbf{x}}_t - \tilde{\mathbf{x}}_i)\boldsymbol{\Lambda}^{-1} \tag{15}$$

where $\boldsymbol{\Lambda} = \text{diag}[l_1^2, \dots, l_{D+F}^2]$.

Similarly the variance of the derivative GP $\mathcal{M}'$ is given as

$$\mathbb{V}_{\mathcal{M}'}[\boldsymbol{\Delta}'_t] = \mathbf{H} - \mathbf{J}^T\mathbf{K}^{-1}\mathbf{J}, \tag{16}$$

where $\mathbf{H}$ is the Hessian matrix containing partial second derivatives of the kernel function w.r.t. the input variables. Note that for computing the derivatives, generally, we need to consider the cross-covariances between the original predictions Δ_t and the derivative predictions Δ'_t (contained in the Jacobian), as well as the covariances between elements of the derivative predictions themselves (contained in the Hessian) – the computation of which can be quite involved (see [17] for details). Fortunately, for our case the computation of the Hessian reduces to $\mathbf{H} = \sigma_f^2\boldsymbol{\Lambda}^{-1}$.

Finally, analogously to Eq. (12), the predictive posterior for the partial derivatives is Gaussian, i.e.

$$\begin{aligned} &p_{\mathcal{M}'}\left(\frac{\partial \mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t]}{\partial \tilde{\mathbf{x}}_t}\middle|\ \tilde{\mathbf{x}}_t\right) \\ &= \mathcal{N}\left(\frac{\partial \mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t]}{\partial \tilde{\mathbf{x}}_t}\middle|\ \mathbf{I} + \mathbb{E}_{\mathcal{M}'}[\boldsymbol{\Delta}'_t], \mathbb{V}_{\mathcal{M}'}[\boldsymbol{\Delta}'_t]\right). \end{aligned} \tag{17}$$

The identity matrix $\mathbf{I}$ above is included since we need to calculate derivatives for the *actual* dynamics function, and not the GP target function which maps to state differences. Hence, we need to include our implicit prior in the derivative. The partial derivatives $f_{\mathbf{x}t}$ and $f_{\mathbf{u}t}$ can then readily be approximated as the mean of this posterior

$$\begin{bmatrix} f_{\mathbf{x}t} \\ f_{\mathbf{u}t} \end{bmatrix} \approx \mathbb{E}\left[\frac{\partial \mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t]}{\partial \tilde{\mathbf{x}}_t}\right] = \mathbf{I} + \mathbb{E}_{\mathcal{M}'}[\boldsymbol{\Delta}'_t]. \tag{18}$$

2) Using the GP Model with iLQR: With these definitions in place iLQR can now be used in conjunction with the GP model $\mathcal{M}$ by approximating the system dynamics f in the *forward*-pass using the GP mean prediction from Eq. (13) as well as replacing the system derivatives $f_{\mathbf{x}t}$ and $f_{\mathbf{u}t}$ in the recursive (*backward*) estimation of the locally optimal controls with the GP mean prediction from Eq. (18). It is important to note that this strategy does *not* take the variance of the prediction into account. It is hence susceptible to severe model bias, as using only the mean functions results in overconfident predictions in areas of the state space where the posteriors from Eq. (12) and Eq. (17) have high variance. While we propose to exploit the uncertainty of the GP as guidance for exploration in Section III-C, using uncertainty directly

within the trajectory optimization algorithm could lead to a more principled formulation. We discuss possible extensions in Section V.

B. Approximate GPR based iLQR

We ultimately aim to use our algorithm for solving control tasks on real world systems with small targeted control intervals δ_t (with typical values $\delta_t \leq 0.05s$ for the systems we target). In such a scenario we have to ensure that the algorithm runtime stays well below δ_t. The bulk of this time will be spent computing the iLQR optimization. Each receding horizon control step requires 1-2 iLQR iterations. For N iterations, $T * (2N + 1)$ calls to the model are required (trajectory length T multiplied by the number of required forward and backward passes). Given that each such call is in in $\mathcal{O}(N)$ (i.e. linear in the number of acquired training points, if the inverse $(\mathbf{K} + \sigma_w^2 \mathbf{I})^{-1}$ is pre-computed), these queries quickly become the bottleneck of the algorithm – necessitating two approximations to make real-time execution possible.

1) Approximate GPR: First, to enable fast computation of the learned forward dynamics (and derivatives) we use a sparse approximation to the trained GP model by employing the Subset of Regressors (SoR) approximation as proposed by [18], [19] (we refer the reader to [20, chapter 8] for a detailed derivation). Concretely, given the GP model $\mathcal{M}$ trained on data $\mathbf{x}_{1\dots N}$ (i.e. for which hyperparameters were already optimized) we compute an approximation to the covariance matrix $\mathbf{K}$ using a set of M reference points $\tilde{\mathbf{x}}^R_{1\dots M} = [\tilde{\mathbf{x}}^R_1, \dots, \tilde{\mathbf{x}}^R_M]$ as:

$$\mathbf{K} \approx \mathbf{K}_R \hat{\mathbf{K}} \mathbf{K}_R^T, \tag{19}$$

where $\mathbf{K}_R \in \mathbb{R}^{N \times M}$ is the covariance matrix between training inputs $\tilde{\mathbf{x}}_{1\dots N}$ and reference points $\tilde{\mathbf{x}}^R_{1\dots M}$ and $\hat{\mathbf{K}} \in \mathbb{R}^{M \times M}$ is the covariance matrix between reference points. This yields an approximation $\hat{m}_{\mathcal{M}}(\tilde{\mathbf{x}})$ to the true posterior mean $\mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t]$ from Eq. (10),

$$\mathbb{E}_{\mathcal{M}}[\boldsymbol{\Delta}_t] \approx \hat{m}_{\mathcal{M}}(\tilde{\mathbf{x}}) = \mathbf{k}_{R^*}^T (\mathbf{K}_R^T \mathbf{K}_R + \sigma_{\mathbf{w}}^2 \hat{\mathbf{K}})^{-1} \mathbf{K}_R^T \mathbf{y}, \tag{20}$$

with $\mathbf{k}_{R^*}^T = [k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}^R_1), \dots, k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}^R_M)]$, as well as an approximation $\hat{\mathbb{V}}$ to the posterior variance $\mathbb{V}$,

$$\mathbb{V}_{\mathcal{M}}[\boldsymbol{\Delta}_t] \approx \hat{\mathbb{V}}[\boldsymbol{\Delta}_t] = \sigma_{\mathbf{w}}^2 \mathbf{k}_{R^*}^T (\mathbf{K}_R^T \mathbf{K}_R + \sigma_{\mathbf{w}}^2 \hat{\mathbf{K}})^{-1} \mathbf{k}_{R^*}. \tag{21}$$

Analogously to Eq. (14) and Eq. (17) we can also compute an approximate derivative GP with mean

$$\mathbb{E}_{\mathcal{M}'}[\boldsymbol{\Delta}'_t] \approx \hat{\mathbb{E}}_{\mathcal{M}'}[\boldsymbol{\Delta}'_t] = \hat{\mathbf{J}}^T (\mathbf{K}_R^T \mathbf{K}_R + \sigma_{\mathbf{w}}^2 \hat{\mathbf{K}})^{-1} \mathbf{K}_R^T \mathbf{y}, \tag{22}$$

and variance

$$\mathbb{V}_{\mathcal{M}'}[\boldsymbol{\Delta}'_t] \approx \hat{\mathbb{V}}_{\mathcal{M}'}[\boldsymbol{\Delta}'_t] = \sigma_{\mathbf{w}}^2 \hat{\mathbf{J}}^T (\mathbf{K}_R^T \mathbf{K}_R + \sigma_{\mathbf{w}}^2 \hat{\mathbf{K}})^{-1} \hat{\mathbf{J}}, \tag{23}$$

where $\hat{\mathbf{J}}$ is the Jacobian of the covariance function between $\tilde{\mathbf{x}}_t$ and the reference points calculated identically to Eq. (15).

To use this approximation in practice it remains to specify how the M reference points are selected. While we could, in principle, select a subset of the N training points (or use M clusters extracted among them) we found that randomly sampling reference points according to the length scales $\boldsymbol{\Lambda}$ performed equally well in our experiments:

$$\tilde{\mathbf{x}}^R_{1:M} = [\tilde{\mathbf{x}}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda}) | \forall i \in [1, M]]. \tag{24}$$

This has two advantages. First, random sampling prevents over-representing common regions of the state space. Second, the sampling approach is independent of the training data, enabling efficient online updates to the GP posterior without re-selection of the reference points. It should be noted that this choice might be unsuitable for even more complex problems.

2) One-Step-Ahead Receding Horizon Planning: Second, to increase the overall time available for computing the iLQR sequence we propose to (asynchronously) pre-compute the receding horizon controls $\mathbf{u}^*_{1:T}$ for the next state $\mathbf{x}_{t+1}$ in state $\mathbf{x}_t$. This is possible due to the fact that given a control sequence $\bar{\mathbf{u}}_{1:T}$ – i.e. the controls that were pre-computed in the previous step using the same procedure we will now outline – and corresponding $\bar{\mathbf{x}}_{1:T}$ starting in $\mathbf{x}_t$ (i.e. $\bar{\mathbf{x}}_1 \approx \mathbf{x}_t$) we can simply compute the prediction

$$\mathbf{x}_{t+1} \approx \hat{\mathbf{x}}_{t+1} = \mathbf{x}_t + \hat{m}_{\mathcal{M}} \left(\begin{bmatrix} \mathbf{x}_t \\ \bar{\mathbf{u}}_1 \end{bmatrix} \right) \tag{25}$$

from the measured $\mathbf{x}_t$ according to Eq. (20). We can then proceed with the standard receding-horizon iLQR optimization warm-started with time-shifted controls (and corresponding time-shifted reference trajectory)

$$\overleftarrow{\mathbf{u}}_{1:T} = [\bar{\mathbf{u}}_2, \dots, \bar{\mathbf{u}}_T, \bar{\mathbf{u}}_T] \tag{26}$$

$$\overleftarrow{\mathbf{x}}_{1:T} = [\hat{\mathbf{x}}_2, \bar{\mathbf{x}}_3, \dots, \bar{\mathbf{x}}_T, \hat{\mathbf{x}}_{T+1}], \tag{27}$$

where $\hat{\mathbf{x}}_{T+1} = \bar{\mathbf{x}}_T + \hat{m}_{\mathcal{M}}([\bar{\mathbf{x}}_T, \bar{\mathbf{u}}_T]^T)$ and $\hat{\mathbf{x}}_2 = \hat{\mathbf{x}}_{t+1}$ is the prediction from (25). This then yields the desired controls $\hat{\mathbf{u}}^*_{1:T}$. This one-step-ahead planning can be executed asynchronously while waiting for δ_t in order to observe the current actions effect.

Once δ_t time has passed $\mathbf{u}^*_1$ is a valid control signal for the measured state $\mathbf{x}_{t+1}$ and can be applied immediately. However, recall that $\mathbf{u}^*_1$ does not yet include a correction term for differences between the prediction $\hat{\mathbf{x}}_{t+1}$ and $\mathbf{x}_{t+1}$. Executing it would therefore correspond to open-loop control (including model errors), which is problematic for systems requiring high accuracy control. Assuming that the learned model is reasonably accurate (e.g. $\|\hat{\mathbf{x}}_{t+1} - \mathbf{x}_{t+1}\|_2 < \epsilon$) we can, nonetheless, achieve closed loop control by computing a corrected action $\mathbf{u}$ using gains $\mathbf{g}^*_1$ and $\mathbf{G}^*_1$ according to Eq. (8), i.e.,

$$\mathbf{u} \leftarrow \hat{\mathbf{u}}^*_1 + \mathbf{g}^*_1 + \mathbf{G}^*_1 (\mathbf{x}_{t+1} - \hat{\mathbf{x}}_{t+1}). \tag{28}$$

C. Variance-Based Exploration

When model-based iLQR (or model-based RL in general) is applied to a real system without incorporating prior knowledge about the system dynamics – that is, no informative prior over models is used and the model is learned using a general function approximator – special care has to be taken to avoid bias in the optimization due to overconfident predictions. Such predictions can occur when the policy visits parts of the state space that are not sufficiently described by the training examples. In that case the model predictions can become arbitrarily bad and the greedy iLQR optimization fails, as it assumes perfect accuracy of the model (and does not take uncertainty into account).

To counter this model-bias we propose to use the variance of the GP posterior to drive exploration of the state space. In a nutshell, we augment the reward function r with a term

rewarding actions that cause high-variance predictions; and hence, when sampled, will improve the model. Formally, we propose to replace the original reward function $r(\mathbf{x}_t, \mathbf{u}_t)$ by augmented rewards

$$\tilde{r}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \lambda \hat{\mathbb{V}}[\boldsymbol{\Delta}_t], \tag{29}$$

where λ is a weighting factor trading off real rewards with exploration. Since iLQR requires quadratic rewards the second term – corresponding to the prediction variance – $\hat{\mathbb{V}}[\boldsymbol{\Delta}_t]$ in (29) must be approximated around the reference trajectory $\bar{\mathbf{x}}_{1:T}$ using a Taylor expansion,

$$\begin{aligned}\hat{\mathbb{V}}[\boldsymbol{\Delta}_t] \approx \hat{\mathbb{V}}[\bar{\boldsymbol{\Delta}}_t] &+ \delta_{\bar{\mathbf{x}}_t}^T \frac{\partial \hat{\mathbb{V}}[\boldsymbol{\Delta}_t]}{\partial \mathbf{x}_t} + \delta_{\bar{\mathbf{u}}_t}^T \frac{\partial \hat{\mathbb{V}}[\boldsymbol{\Delta}_t]}{\partial \mathbf{u}_t} \\ &+ \frac{1}{2}\delta_{\bar{\mathbf{x}}_t}^T \frac{\partial \hat{\mathbb{V}}[\boldsymbol{\Delta}_t]}{\partial \mathbf{x}_t \mathbf{x}_t}\delta_{\bar{\mathbf{x}}_t} + \frac{1}{2}\delta_{\bar{\mathbf{u}}_t}^T \frac{\partial \hat{\mathbb{V}}[\boldsymbol{\Delta}_t]}{\partial \mathbf{u}_t \mathbf{u}_t}\delta_{\bar{\mathbf{u}}_t} \\ &+ \delta_{\bar{\mathbf{x}}_t}^T \frac{\partial \hat{\mathbb{V}}[\boldsymbol{\Delta}_t]}{\partial \mathbf{x}_t \mathbf{u}_t}\delta_{\bar{\mathbf{u}}_t}\end{aligned} \tag{30}$$

This is analogous to the approximation of the original reward function in Eq. (5) and can be computed analytically using the terms from Eq. (23).

D. Combined Algorithm

The algorithm for the complete method which we dub *Approximate iLQR with GPs* (AGP-iLQR) implementing the previously described details is given in Algorithm 1. As listed we perform online model learning, concurrently updating the model during execution. In practice we interleave episodes of AGP-iLQR with optimization of the GP hyperparameters, where we iteratively maximize the marginal log-likelihood of the GP observations using conjugate gradient descent. For the algorithm to perform as advertised the concurrent computation of the one-step-ahead controls in line 12 must be completed before the next action execution (line 6). This is in general not guaranteed – although for the typical control interval we encounter in the following experiments ($\delta_t = 0.05s$) this was always the case. To ensure fail-safe operation one could circumvent this drawback in practice by executing a default action (e.g. $\mathbf{u} = \mathbf{0}$) when the concurrent execution does not finish in time.

Algorithm 1 Approximate iLQR with GPs (AGP-iLQR)

Input: initial state $\mathbf{x}_1$; number of iLQR iterations s; receding-horizon length T; episode length E (or $E = \infty$ for true online control); GP hyperparameters $\boldsymbol{\Lambda}$, σ_f^2, $\sigma_{\mathbf{w}}^2$; reward function $r(\mathbf{x}_t, \mathbf{u}_t)$; initial controls $\mathbf{u}_1$ with corresponding $\mathbf{g}_1$ and $\mathbf{G}_1$ (e.g. $\mathbf{u}_1 = \mathbf{g}_1 = \mathbf{G}_1 = \mathbf{0}$)

1: $\tilde{\mathbf{x}}_{1:M}^R \leftarrow$ Sample as $[\tilde{\mathbf{x}}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda}) | \forall i \in [1, M]]$
2: $\mathbf{u}_{1:T}^* \leftarrow [\mathbf{u}_1, \mathbf{0}, \ldots, \mathbf{0}]$
3: $\mathbf{x}_1^* \leftarrow \mathbf{x}_1$
4: **for** $t \leftarrow 1$ **to** E **do**
5: Get measurement from system: $\mathbf{x}_t$
6: Execute: $\mathbf{u} \leftarrow \mathbf{u}_1^* + \mathbf{g}_1^* + \mathbf{G}_1^*(\mathbf{x}_t - \mathbf{x}_1^*)$
7: **concurrently do**
8: $\boldsymbol{\Delta}_t \leftarrow (\mathbf{x}_{t-1} - \mathbf{x}_t)$
9: $\mathcal{M} \leftarrow$ update model with $(\tilde{\mathbf{x}}_{t-1}, \boldsymbol{\Delta}_t)$
10: Predict: $\hat{\mathbf{x}}_2 \leftarrow \mathbf{x}_t + \tilde{\mathbb{E}}_{\mathcal{M}}[\boldsymbol{\Delta}_t]$
11: $\overleftarrow{\mathbf{u}}_{1:T} \leftarrow [\mathbf{u}_2^*, \ldots \mathbf{u}_T^*, \mathbf{u}_T^*]$
12: Solve for one-step-ahead controls with iLQR (using Eqs. (6-8)):

$$(\mathbf{u}^*, \mathbf{x}^*)_{1:T} \in \arg\max_{\mathbf{u}_{1:T}, \mathbf{x}_{1:T}} \mathbb{E}_{\mathcal{M}}[\sum_{t=1}^{T} \tilde{r}(\mathbf{x}_t, \mathbf{u}_t) \mid \hat{\mathbf{x}}_2, \overleftarrow{\mathbf{u}}_{1:T}]$$
$$\text{with } \tilde{r}(\mathbf{u}_t, \mathbf{x}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \lambda \tilde{\mathbb{V}}[\boldsymbol{\Delta}_t]$$

13: **end**
14: **end**

IV. Experiments

We evaluate our algorithm on several benchmark problems in simulation (two-link manipulator, cart-pole swing-up) and on a real system (cart-pole swing-up). We consistently used a quadratic reward function based on the distance to a pre-defined goal state (with optional augmentation for variance-based exploration) in all our experiments:

$$r(\mathbf{x}_t, \mathbf{u}_t) = (\mathbf{x}^G - \mathbf{x}_t)^T D (\mathbf{x}^G - \mathbf{x}_t) + \mathbf{u}_t^T H \mathbf{u}_t. \tag{31}$$

A. Two-Link Manipulator

To check the validity of our approach, we performed a preliminary experiment with the two-link manipulator which simulates a simple robotic arm consisting of two actuated rigid links in a horizontal plane. The manipulator has to be controlled towards a goal configuration from arbitrary initial configurations. The state vector $\mathbf{x}$ is hence 4-dimensional (the angle of both links as well as their velocities)[1]. The manipulator is controlled by setting torques for the motors between the two links (i.e. we have 2-dimensional actions $\mathbf{u}$). For the exact physical properties we refer to Busoniu et al. [21].

We set the goal $\mathbf{x}_G = [0, 0, 0, 0]^T$ (i.e. the arm is in a fully stretched resting position, pointing upwards) use reward weighting terms $D = \text{diag}[-10, -1, -10, -1]$ and $H = \text{diag}[-1, -1]$ and used a set of 25 fixed random starting configurations in the interval $[-\pi, \pi)$ for testing after each learning episode. Since this experiment was designed as an initial test we evaluated the most basic configuration of our algorithm – using the full GP Model (without approximation) and not enabling variance-based exploration. We compare this to receding horizon iLQR using the true model of the simulation. We used an episode length of $E = 120$ and a horizon of $T = 10$ in both settings. Hyperparameters of the GP were optimized for 100 steps between episodes.

The results of running this experiment are depicted in Figure 1. As expected, in the beginning of the experiment our approach performed worse than receding horizon iLQR using the real dynamics (RH-iLQR true model) but quickly found a good model of the system, catching up to the performance of RH-iLQR after a single episode (or 120 steps) of interacting with the system.

B. Simulated Cart-Pole Swing-Up

We next turn to the cart-pole system, which consists of a cart to which a free swinging pole is attached. The cart is fixed

[1]To circumvent problems with non discontinuities at angle positions $\pm\pi$ we expand the state to a complex representation of the angles before feeding the extended state to the GP.

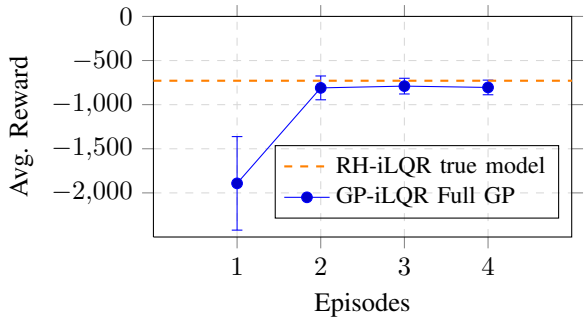

Fig. 1: Evaluation of GP based iLQR, on the Two-Link Manipulator averaged over a fixed set of 25 randomly sampled starting positions. We also plot the performance of using the true dynamics model as a baseline.

on a rail on which it can freely move left and right. Starting from a configuration where the pole is facing downwards, the goal of the swing-up task is to swing it upwards and balance it with the cart centered on the rail. The state $\mathbf{x}_t$ of the system is modeled using four continuous variables: the angle of the pole θ (which, as before, is represented using sine and cosine components), its angular velocity $\dot{\theta}$, the cart position p and the cart velocity $\dot{p}$. The continuous one-dimensional action u_t determines motor force. We implemented the same differential equations and used the same parameter settings as in [22], deviating only in one regard: setting the control interval to $\delta_t = 0.05$ rather than using $\delta_t = 0.1$. This ensures both that local linearization of the dynamics succeeds and that the control interval is closer to the one of the real cart-pole in the next experiment (facilitating transfer of hyperparameter settings).

We assess the influence of several parameters of our algorithm as well as its overall ability to cope with a more difficult control problem on this benchmark. We used reward weighting terms $D = \text{diag}[-40, 0, -80, 0]$ and $H = \text{diag}[-1]$ in our experiments. Episode length was 150 control cycles, and no variance based exploration was used. Hyperparameters of the GP were optimized for 100 steps between each episode. All results are averaged over 25 runs, accounting for variance due to the randomly chosen first action as well as random sampling of the reference points.

1) Influence of GP approximation: Fig. 2 shows the performance on the cart-pole using a varying number of reference points M together with two baselines (iLQR using the true model, as well as performance using the full GP model). The planning horizon was fixed at 15. As for the Two-Link experiment our algorithm using the full GP model quickly learned a good model, reached the performance of iLQR based on the true system model and successfully learned a swing-up behavior after 2-3 episodes (Avg. Reward > -13000). Introducing the sparse approximation did not result in a performance decrease – but rather stabilized performance in the first two iterations – even when only a moderate amount of 100-150 reference points were used. Further decreasing the number of reference points to $M = 50$ then resulted in sub-optimal trajectories. We also experimented with using a larger number (200-250) reference points which, however, did not result in a significant improvement.

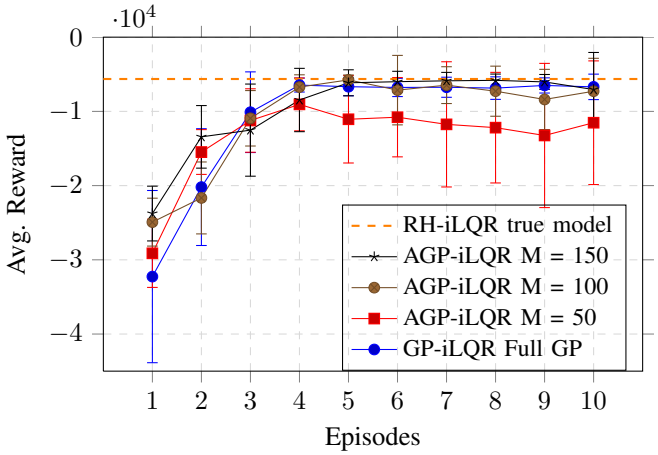

Fig. 2: Evaluation of AGP-iLQR, on the simulated cart-pole, using varying number of reference points M in the GP approximation. Results are averaged over 25 runs. As baselines we also plot the performance of using the full GP model without approximation, and of iLQR using the true model.

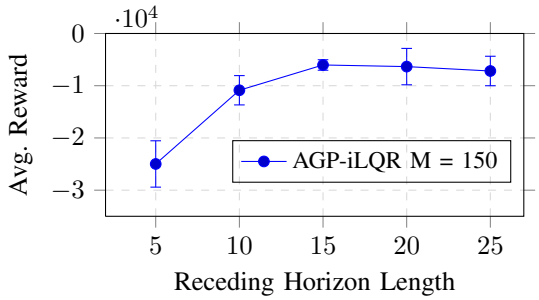

Fig. 3: Performance on the simulated cart-pole with changing receding horizon length after 10 episodes of training. Results are averaged over 25 evaluations. M = 150 reference points were sampled for each evaluation.

2) Influence of planning horizon: Fixing the number of reference points to $M = 150$, Fig. 3 shows the influence of varying receding horizon lengths (combined with two iLQR iterations for each setting). It shows that choosing a short horizon (< 10) resulted in a controller that fails to perform the task. Furthermore, increasing the horizon beyond 15 did not result in better controllers and we hence fixed it to 15 in further experiments.

3) Overall performance: Overall the best configuration of our algorithm, using 150 reference points and a planning horizon of 15, learned a successful swing-up and balance controller in 2-3 episodes or 12-18 seconds of interaction with the system (corresponding to 240-360 transition samples). It is thus similarly data-efficient as other recent work on using GP models for control [22], [2]. Notably, in this (noise-free) simulation setting we did not encounter problems with model-bias despite using only the GP posterior mean.

C. Real Cart-Pole Swing-Up

As a final experiment we applied the proposed architecture to a real cart-pole system as depicted in Fig. 4. The real system we used in our experiments consists of a cart with mass of

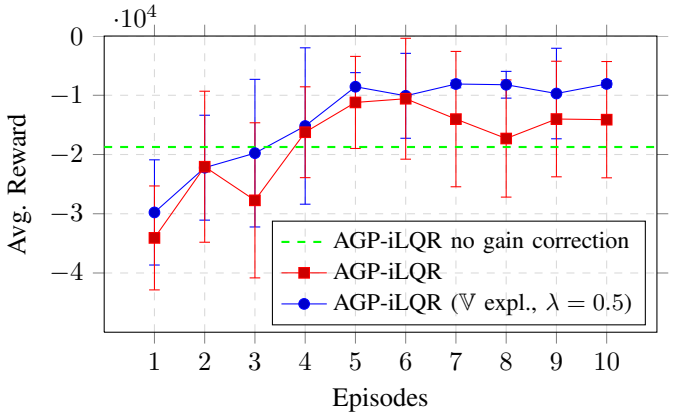

Fig. 5: Reward achieved by AGP-iLQR, with and without variance based exploration ($\mathbb{V}$ expl.), when evaluated on the real cart-pole system. Results are averaged over 10 trials.

approximately 1kg and a pole with mass of approximately 0.1kg and length 0.38m. The 1-dimensional action in this setup directly controls the voltage of a motor driving the cart and can be set between -10 and 10 volts. We set the maximum length of each episode to 150 control cycles. Due to the limited length of the rail in the real setup the cart cannot move further than 0.3m to each side. The swing-up problem on this system is thus more complicated than the one we considered in simulation; as two swings are necessary to bring the pole upright. Further, for the task to succeed, the control interval may not exceed 0.05s which drastically limits computation time. To circumvent this problem we used an approximated GP model with 150 reference points in combination with Predictive Receding Horizon Planning (see Sections III-B2 and III-B1). As before the hyperparameters of the GP were optimized for 100 steps between each episode.

The results of running AGP-iLQR with a receding horizon of 15 on this system are shown in Fig. 5. Note that we do not have a high accuracy ground truth model of the system and hence cannot apply iLQR to this problem without our learning algorithm. As can be seen, both the version with variance-based exploration as well as the version without quickly learned a good model of the environment. Qualitatively (not shown in plot) both managed to achieve successful swing-up and balancing trajectories after 4-7 episodes. However, when variance-based exploration was not used, the iLQR optimization became overly optimistic (i.e. model learning often failed to sufficiently reduce model uncertainty about the dynamics). Since the iLQR optimization itself is agnostic to model uncertainty, this results in sub-optimal controllers and thus in higher variance in the Figure. We also plot the effect of removing the local gain correction (Line 6 in Algorithm 1) after learning a successful controller. We observed that with this change performance degraded, resulting in a controller that cannot reliably balance the pole upright, due to overconfidence in the one-step-ahead prediction. As a control we also experimented with executing a pre-planned trajectory of length 150 based (i.e. not using the receding horizon scheme) on the model learned with AGP-iLQR which, however, did not result in a successful swing-up behavior. This indicates that one-step-ahead receding horizon control is a crucial ingredient for success on this real system. A successful trajectory executed by our algorithm is depicted in Fig. 4 a video of a complete learning run can be found at http://goo.gl/o4j8sX. A summary of our algorithms performance is given in Table I.

TABLE I: Required number of episodes for AGP-iLQR until a successful trajectory for the three considered benchmarks.

Domain	# Episodes	Time	# samples
Two-Link	1	6 s	120
Cart-Pole Sim.	2-3	12 - 18 s	240-360
Cart-Pole Real	4-7	20 - 42 s	480-840

V. Discussion and Related Work

Our approach is related to a large body of work on both trajectory optimization with (stochastic) optimal control and dynamics model learning for robotics. In the following we give a brief overview of its connection to existing work as well as possibilities for improvements.

For performing online trajectory optimization we employ a receding horizon variant of the iterative LQR (iLQR) algorithm [5], [14]. In contrast to standard differential dynamic programming [12] iLQR does not use second-order information of the dynamics function. As calculating second order information is usually costly this is advantageous in a real-world control setting. The key difference of our algorithm compared to standard iLQR is that we do not assume the availability of a true system model but rather learn a non-parametric model from data. Furthermore, differing from [14] we employ a one step ahead predictive control scheme – using the learned model – which enables real-time control.

Combining iLQR with a learned model is a well established research problem in the control and reinforcement learning communities. An early approach was presented by Schaal and Atkeson [23]. They explored learning of models of a robot arm for offline trajectory optimization with DDP. They however encountered situations were learning fails due to model bias as they neither employed online learning nor receding horizon planning. More recently Mitrovic et al. [24] combined iLQR with an adaptive model based on Locally Weighted Projection Regression. Similar to the work we present, they use analytic derivatives of the learned model w.r.t the input variables for the iLQR optimization steps. However as in [23] they did not use a receding horizon scheme for online re-planning which makes their approach inapplicable for the control problems we consider (cf. Section IV-C).

Among the attempts to combine DDP with a learned model the work closest to ours is that of Coates and colleagues [1]. They identify a model for an autonomous helicopter and use it to optimize a feedback controller using online receding horizon DDP [1]. Their algorithm for model identification [25] is very efficient but requires considerable expert knowledge about the system to be controlled. In contrast, our GP-based model learning algorithm is computationally more expensive, but relies on less prior knowledge and provides estimates about the uncertainty of its predictions.

For model learning we build on existing work for employing Gaussian Processes Regression in order to estimate

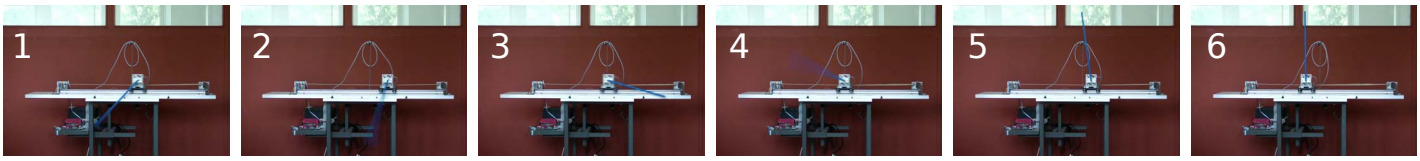

Fig. 4: Depiction of a successful trajectory on the real cart-pole. Two swings are necessary to swing-up the pole on this system. We only show the last swing due to space constraints. We refer to the video (http://goo.gl/o4j8sX) for a full run of the algorithm.

the transition function of the dynamical system: Kuss and Rasmussen [26] use a GP dynamics model in combination with a GP based value function approximation for reinforcement learning. GPs have recently also been used for (inverse) dynamics model learning in robotics (see e.g. [15], [16]). The formulation we present in this paper is directly based on recent work by Deisenroth et al. [2], where the authors use GPs to learn the dynamics, and estimate expected long term rewards in a policy-search approach based on these models.

Augmenting the reward function with a variance-based term has previously been considered e.g. in [27] and [28].

Despite the computational efficiency of using the instantaneous variance of our GP predictions it would be desirable to incorporate the GP uncertainty into the trajectory optimization in a more principled manner. One possible way to achieve this would be to integrate our GP-based model in the Approximate Inference Control (AICO) [6] framework. AICO casts the trajectory optimization problem as one of probabilistic inference, which uses local messages passed between random variables along a trajectory. This would be a natural fit for the probabilities of state transitions output by our GP models.

VI. Conclusion

We presented AGP-iLQR, an algorithm for joint model learning and optimal control of non-linear dynamical systems based on Gaussian Process models and an adapted iLQR formulation. It achieves real-time control capability through (1) asynchronous, predictive, one-step-ahead planning and (2) a sparse approximation of the GP. It addresses model-bias through variance-based exploration. Our experiments show that the presented algorithm is highly data-efficient, learning to control three benchmark problems in few system interactions. Most notably our algorithm successfully learned to control a real cart-pole system with unknown dynamics from scratch.

Acknowledgments

This work was supported by DFG project number RI923/7-1. Support also came from the BrainLinks-BrainTools cluster of excellence, DFG-EX-C1086. We thank Christof Schoetz for help with the implementation, Thomas Lampe for help with the real cart-pole system, Manuel Blum for developing libgp, and the three anonymous reviewers for helpful comments.

References

[1] A. Coates, P. Abbeel, and A. Y. Ng, "Learning for Control from Multiple Demonstrations," in *ICML*, 2008.

[2] M. Deisenroth and C. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search," in *ICML*, 2011.

[3] R. Hafner and M. Riedmiller, "Reinforcement Learning in Feedback Control," *Machine Learning*, 2011.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey," *IJRR*, 2013.

[5] W. Li and E. Todorov, "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems," in *ICINCO (1)*, 2004, pp. 222–229.

[6] M. Toussaint, "Robot Trajectory Optimization using Approximate Inference," in *ICML*, 2009.

[7] S. Levine and V. Koltun, "Variational Policy Search via Trajectory Optimization," in *NIPS*, 2013.

[8] ——, "Learning Complex Neural Network Policies with Trajectory Optimization," in *ICML*, 2014.

[9] D. Nguyen-Tuong and J. Peters, "Model Learning for Robot Control: A Survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, 2011.

[10] S. Schaal and C. Atkeson, "Learning Control in Robotics," *Robotics Automation Magazine, IEEE*, vol. 17, no. 2, pp. 20–29, June 2010.

[11] R. I. Brafman and M. Tennenholtz, "R-max - A General Polynomial Time Algorithm for Near-optimal Reinforcement Learning," *JMLR*, vol. 3, pp. 213–231, 2003.

[12] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*. New York: Elsevier, 1970.

[13] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and Stabilization of Complex Behaviors Through Online Trajectory Optimization," in *IROS*, 2012.

[14] Y. Tassa, T. Erez, and W. D. Smart, "Receding Horizon Differential Dynamic Programming," in *NIPS*, 2008.

[15] C. Hartmann, J. Boedecker, O. Obst, S. Ikemoto, and M. Asada, "Real-Time Inverse Dynamics Learning for Musculoskeletal Robots based on Echo State Gaussian Process Regression," in *R:SS*. MIT Press, 2013.

[16] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model Learning with Local Gaussian Process Regression," *Advanced Robotics*, 2009.

[17] E. Solak, R. Murray-smith, W. E. Leithead, D. J. Leith, and C. E. Rasmussen, "Derivative Observations in Gaussian Process Models of Dynamic Systems," in *NIPS*, 2003.

[18] G. Wahba, *Spline Models for Observational Data*. SIAM [Society for Industrial and Applied Mathematics], 1990.

[19] T. Poggio and F. Girosi, "Networks for Approximation and Learning," *Proceedings of the IEEE*, 1990.

[20] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006.

[21] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 2010.

[22] M. P. Deisenroth, "Efficient Reinforcement Learning using Gaussian Processes," Ph.D. dissertation, 2010.

[23] C. G. Atkeson and S. Schaal, "Robot Learning from Demonstration," in *ICML*, 1997, pp. 12–20.

[24] D. Mitrovic, S. Klanke, and S. Vijayakumar, "Optimal Control with Adaptive Internal Dynamics Model," in *ICINCO*, 2008.

[25] P. Abbeel, V. Ganapathi, and A. Y. Ng, "Learning Vehicular Dynamics, with Application to Modeling Helicopters," in *NIPS*, 2006.

[26] M. Kuss and C. E. Rasmussen, "Gaussian Processes in Reinforcement Learning," in *NIPS*, 2004.

[27] A. Simpkins, R. De Callafon, and E. Todorov, "Optimal Trade-Off Between Exploration and Exploitation," in *Proceedings of the American Control Conference*, 2008.

[28] J. Sorg, S. P. Singh, and R. L. Lewis, "Variance-Based Rewards for Approximate Bayesian Reinforcement Learning," in *UAI*, 2010.