2015 American Control Conference
Palmer House Hilton
July 1-3, 2015. Chicago, IL, USA

Data-Driven Differential Dynamic Programming Using Gaussian Processes

Yunpeng Pan and Evangelos A. Theodorou

Abstract— **We present a Bayesian nonparametric trajectory optimization framework for systems with unknown dynamics using Gaussian Processes (GPs), called Gaussian Process Differential Dynamic Programming (GPDDP). Rooted in the Dynamic Programming principle and second-order local approximations of the value function, GPDDP learns time-varying optimal control policies from sampled data. Based on this framework, we propose two algorithms for implementations. We demonstrate the effectiveness and efficiency of the proposed framework using three numerical examples.**

I. Introduction

Differential dynamic programming (DDP) is a powerful trajectory optimization approach. Originally introduced in [1], DDP generates both open and closed-loop optimal control policies along with an optimal state trajectory. Compared with global optimal control approaches, the local optimal DDP shows superior applicability to high-dimensional problems. Over the last decade, variations of DDP have been proposed in both control and machine learning communities [2], [3], [4], [5], [6], [7]. A recent study demonstrated the applicability of DDP to high-dimensional robotic systems [8].

DDP is derived based on linear approximations of the nonlinear dynamics along state and control trajectories, therefore it relies on accurate and explicit dynamics models. However, modeling a dynamical system is generally a challenging task and model uncertainty is one of the principal limitations of model-based trajectory optimization methods. Various approaches have been developed to address these issues. In [9], a minimax criterion of DDP was employed to correct modeling errors and attenuate unknown disturbances. In [10], a complex robotic system was parameterized and learned from expert-demonstrated trajectories. Despite the broad applicability of DDP based on either parametrically learned or a-priori known dynamics, in general the interdependencies between various elements (e.g., electromechanical, pneumatic, hydraulic) of autonomous systems are far from being easily represented by deterministic models. To overcome this limitation, probabilistic models, such as Gaussian processes (GPs), have increasingly drawn more attention in control and machine learning communities.

GPs are Bayesian nonparametric models such that the model uncertainty can be quantified explicitly. GPs are defined as distributions over functions and GP regression operates in the function spaces. Different from typical parametric models, GPs do not assume prior structure of the model and take into account measurement noises inherently.

The GP approach has been widely used for model learning over the last decade since the systematic introduction in [11]. Recent developments in GPs have demonstrated promising applicability in nonlinear filtering, optimal control and reinforcement learning [12], [13], [14], [15], [16], [17]. In this paper, we use GP regression to learn unknown dynamics model, and incorporate GP inference into the DDP framework. The main characteristics of GPDDP can be summarized as follows: 1) GPDDP learns Bayesian nonparametric models from sampled data and iteratively optimizes a trajectory. The probabilistic representation of the dynamics have shown impressive data/sample efficiency in reinforcement learning [17]. 2) GPDDP performs linearization by computing Jacobian matrices analytically, avoids the computational burden of finite differences. 3) GPDDP can be applied in both online and offline settings for different types of applications.

The rest of this paper is organized as follows: Section II provides the basic problem formulation. In Section III, we introduce the GP representation of the dynamics. Section IV derives the explicit form of optimal control policy. In Section V, we propose two algorithms and discuss implementation details. Numerical results are provided and discussed in Section VI. Finally Section VII concludes this paper.

II. Problem formulation

We consider a general unknown nonlinear system described by the following stochastic differential equation

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u})\mathrm{d}t + \mathbf{C}\mathrm{d}\omega, \quad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{1}$$

with state $\mathbf{x} \in \mathbb{R}^n$, control $\mathbf{u} \in \mathbb{R}^m$, and standard Brownian motion noise $\omega \in \mathbb{R}^p$ such that $\mathrm{d}\omega \sim \mathcal{N}(0, \mathbf{\Sigma}_\omega)$. We consider the optimal control problem with cost:

$$J^\pi(\mathbf{x}, t) = \mathbb{E}\bigg[h\Big(\mathbf{x}(t_f)\Big) + \int_{t_0}^{t_f} \mathcal{L}\Big(\mathbf{x}(t), \pi(\mathbf{x}, t), t\Big)dt\bigg], \tag{2}$$

where $h(\mathbf{x}(t_f))$ is the terminal cost, $\mathcal{L}(\mathbf{x}(t), \pi(\mathbf{x}, t), t)$ is the instantaneous cost rate which is a function of state $\mathbf{x}$ and control policy $\pi(\mathbf{x}(t), t)$. The cost $J^\pi(\mathbf{x}, t)$ is defined as the expectation accumulated over the time horizon $(t_0, \ldots, t_f)$ starting from the initial state $\mathbf{x}(t_0)$ to final state $\mathbf{x}(t_f)$. For the rest of our analysis, we denote $\mathbf{x}_k = \mathbf{x}(t_k)$ in discrete-time where $k = 0, 1, ...N$ is the time step, we use this subscript rule for other variables as well.

The authors are with the Autonomous Control and Decision Systems Laboratory, at the Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, 270 Ferst Drive Atlanta GA 30332-0150, USA ypan37@gatech.edu etheodorou3@mail.gatech.edu

III. Dynamics model learning using Gaussian Process Regression

The continuous functional mapping from state-control pair $\tilde{\mathbf{x}} = (\mathbf{x}, \mathbf{u}) \in \mathbb{R}^{n+m}$ to state transition $\mathrm{d}\mathbf{x}$ can be viewed as an inference with the goal of inferring state transition $\mathrm{d}\mathbf{x}$ given $(\mathbf{x}, \mathbf{u})$. We view this inference as a nonlinear regression problem. In this section, we introduce the Gaussian processes (GP) approach to learning dynamical model (1). A GP is defined as a collection of random variables, any finite number subset of which have a joint Gaussian distribution. Given a sequence of state-control pair $\tilde{\mathbf{X}} = \{(\mathbf{x}_0, \mathbf{u}_0), \dots (\mathbf{x}_N, \mathbf{u}_N)\}$, and the corresponding state transition $\mathrm{d}\mathbf{X} = \{\mathrm{d}\mathbf{x}_0, \dots, \mathrm{d}\mathbf{x}_N\}$, a GP is completely defined by a mean function and a covariance function. The joint distribution of the observed state transitions and the transition corresponding to a given test state-control pair $\tilde{\mathbf{x}}^* = (\mathbf{x}^*, \mathbf{u}^*)$ can be written as

$$\mathrm{p}\begin{pmatrix} \mathrm{d}\mathbf{X} \\ \mathrm{d}\mathbf{x}^* \end{pmatrix} \sim \mathcal{N}\Bigg(0, \begin{bmatrix} \mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_n \mathbf{I} & \mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{x}}^*) \\ \mathbf{K}(\tilde{\mathbf{x}}^*, \tilde{\mathbf{X}}) & \mathbf{K}(\tilde{\mathbf{x}}^*, \tilde{\mathbf{x}}^*) \end{bmatrix}\Bigg).$$

The covariance of this multivariate Gaussian distribution is defined via a kernel matrix $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)$. In particular, the Gaussian kernel

$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_s^2 \exp(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \mathbf{W} (\mathbf{x}_i - \mathbf{x}_j)). \quad (3)$$

$\sigma_s, \sigma_n, \mathbf{W}$ are called the hyper-parameters of the GP. The kernel function can be interpreted as a similarity measure of random variables. More specifically, if the training pairs $\tilde{\mathbf{X}}_i$ and $\tilde{\mathbf{X}}_j$ are close to each other in the kernel space, their corresponding state transition $\mathrm{d}\mathbf{x}_i$ and $\mathrm{d}\mathbf{x}_j$ are highly correlated. The posterior distribution, which is also a Gaussian distribution, can be obtained by constraining the joint distribution to contain the output $\mathrm{d}\mathbf{x}^*$ that is close to the observations. Assuming independent outputs (no correlation between each output dimension) and given test input $\tilde{\mathbf{x}}_k = [\mathbf{x}_k, \mathbf{u}_k]$ at time k, the one-step prediction of the dynamics based on GP can be evaluated as $\mathrm{p}(\mathrm{d}\mathbf{x}_k | \tilde{\mathbf{x}}_k) \sim \mathcal{N}(\mathrm{d}\boldsymbol{\mu}_k, \mathrm{d}\boldsymbol{\Sigma}_k)$, where the terms $\mathrm{d}\boldsymbol{\mu}_k$ and $\mathrm{d}\boldsymbol{\Sigma}_k$ are the predictive mean and variance of the state transition. They are specified as

$$\begin{aligned} \mathrm{d}\boldsymbol{\mu}_k &= \mathbb{E}[\mathrm{d}\mathbf{x}_k] = \mathbf{K}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}})(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_n \mathbf{I})^{-1} \mathrm{d}\mathbf{X}, \\ \mathrm{d}\boldsymbol{\Sigma}_k &= \mathrm{Var}[\mathrm{d}\mathbf{x}_k] = \mathbf{K}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_k) - \mathbf{K}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}})(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) \\ &\quad + \sigma_n \mathbf{I})^{-1} \mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{x}}_k). \end{aligned} \quad (4)$$

A toy example of GP regression is shown in Fig. 1.

The state distribution at $t = 1$ is $\mathrm{p}(\mathbf{x}_1) \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ where the state mean and variance are: $\boldsymbol{\mu}_1 = \mathbf{x}_0 + \mathrm{d}\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_1 = \mathrm{d}\boldsymbol{\Sigma}_0$. When propagating the GP-based dynamics over a trajectory of time horizon N, the input state $\mathbf{x}_k$ becomes uncertain with a Gaussian distribution. Here we employ the moment matching approach introduced in [18], [19], [17] to approximate the predictive distribution. Thus the distribution over state transition can be computed as [17]

$$\mathrm{p}(\mathrm{d}\mathbf{x}_k) = \int\int\int \mathrm{p}(f(\mathbf{x}_k, \mathbf{u}_k) | \mathbf{x}_k, \mathbf{u}_k) \mathrm{p}(\mathbf{x}_k, \mathbf{u}_k) \mathrm{d}f \mathrm{d}\mathbf{x}_k \mathrm{d}\mathbf{u}_k.$$

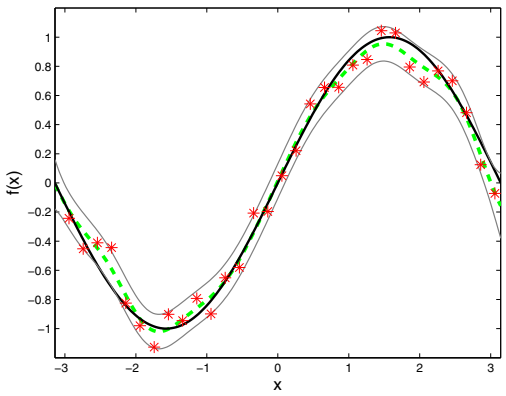


Fig. 1: Simple one-dimensional GP regression, where $\mathbf{f}(\mathbf{x}) = \sin \mathbf{x}$. Red stars are noisy samples drawn from $\mathbf{f}(\mathbf{x})$; the black line is the function $\mathbf{f}(\mathbf{x})$; green dash line is the GP predictive mean, and gray lines represent GP predictive variance.

Generally, the above integration cannot be computed analytically and the nonlinear mapping of input Gaussian distributions lead to non-Gaussian predictive distributions. Based on the moment matching approximation, the state distribution at $k+1$ is a Gaussian [17]

$$\begin{aligned} \boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \mathrm{d}\boldsymbol{\mu}_k, \\ \boldsymbol{\Sigma}_{k+1} &= \boldsymbol{\Sigma}_k + \mathrm{d}\boldsymbol{\Sigma}_k + \mathrm{Cov}[\mathbf{x}_k, \mathrm{d}\mathbf{x}_k] + \mathrm{Cov}[\mathrm{d}\mathbf{x}_k, \mathbf{x}_k]. \end{aligned} \quad (5)$$

The kernel or hyper-parameter $\Theta = (\sigma_n, \sigma_s, \mathbf{W})$ can be learned by maximizing the log-likelihood of the training outputs given the inputs

$$\Theta^* = \underset{\Theta}{\mathrm{argmax}} \left\{ \log \left(\mathrm{p}\Big(\mathrm{d}\mathbf{X} | \tilde{\mathbf{X}}, \Theta\Big) \right) \right\}, \quad (6)$$

where

$$\begin{aligned} \log \left(\mathrm{p}\Big(\mathrm{d}\mathbf{X} | \tilde{\mathbf{X}}, \Theta\Big) \right) &= -\frac{1}{2} \mathrm{d}\mathbf{X}^{\mathrm{T}} \Big(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_n^2 \mathbf{I}\Big)^{-1} \\ \mathrm{d}\mathbf{X} - \frac{1}{2} \log \Big|\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_n^2 \mathbf{I}\Big| - \frac{N}{2} \log 2\pi. \end{aligned} \quad (7)$$

The optimization problem can be solved using numerical methods such as conjugate gradient [11].

IV. Gaussian Process Differential Dynamic Programming

A. Linearization of the GP-based dynamics

As discussed in the last section, the states of the dynamical systems can be approximated by the predictive mean of the GP regression $\boldsymbol{\mu}_k$ in (5). Given a nominal trajectory $(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$ of states and controls, we linearize the dynamics around $(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$. Here we define the state and control deviations $\delta\mathbf{x}_k = \boldsymbol{\mu}_k - \bar{\mathbf{x}}_k, \delta\mathbf{u}_k = \mathbf{u}_k - \bar{\mathbf{u}}_k$ and compute the first-order expansion of the dynamics specified as

$$\mathrm{d}\delta\mathbf{x}_k = \mathbf{f}_{\mu_k} \delta\mathbf{x}_k \mathrm{d}t + \mathbf{f}_{\mathbf{u}_k} \delta\mathbf{u}_k \mathrm{d}t, \quad (8)$$

where $\mathbf{f}_{\mu_k} = \nabla_\mu \mathbf{f}_k$ and $\mathbf{f}_{\mathbf{u}_k} = \nabla_\mathbf{u} \mathbf{f}_k$ are the gradients that can be computed as

$$\nabla_\mu \mathbf{f}_k = \frac{\partial \mathbb{E}(\mathrm{d}\mathbf{x}_k)}{\partial \boldsymbol{\mu}_k}, \qquad \nabla_\mathbf{u} \mathbf{f}_k = \frac{\partial \mathbb{E}(\mathrm{d}\mathbf{x}_k)}{\partial \mathbf{u}_k}. \tag{9}$$

Since $\mathbb{E}(\mathrm{d}\mathbf{x}_k)$ can be represented explicitly in $\boldsymbol{\mu}_k$ and $\mathbf{u}_k$, all partial derivatives can be computed analytically. The linearized dynamics (8) is rewritten in discrete-time form as follows,

$$\delta \mathbf{x}_{k+1} = \underbrace{(I_{n\times n} + \mathbf{f}_{\mu_k} \mathrm{d}t)}_{\mathbf{A}_k} \delta \mathbf{x}_t + \underbrace{(\mathbf{f}_{\mathbf{u}_k} \mathrm{d}t)}_{\mathbf{B}_k} \delta \mathbf{u}_t, \tag{10}$$

After specifying the forms of the state transition and control transition matrices $\mathbf{A}_k$ and $\mathbf{B}_k$, next we discuss how to compute the value function and optimal control policy.

B. Value function approximation

The Bellman equation for the cost-to-go function is defined as

$$V(\mathbf{x}_k, t) = \min_{\mathbf{u}_k} \mathbb{E}\Big[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t + V(\mathbf{x}_{k+1}, k+1)\Big|\mathbf{x}_t\Big]. \tag{11}$$

As in the classical DDP, we expand the cost-to-go function up to the second order around a given nominal trajectory $(\bar{\mathbf{x}},\bar{\mathbf{u}})$. First we expand the running cost:

$$\begin{aligned}\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t &= \mathcal{L}\Big(\bar{\mathbf{x}}_k + \delta\mathbf{x}_k, \bar{\mathbf{u}}_k + \delta\mathbf{u}_k\Big)\mathrm{d}t \\ &= q_k + \mathbf{q}_k^\mathrm{T}\delta\mathbf{x}_k + \mathbf{r}_k^\mathrm{T}\delta\mathbf{u}_k \\ &+ \frac{1}{2}\delta\mathbf{x}_k^\mathrm{T}\mathbf{Q}_k\delta\mathbf{x}_k + \frac{1}{2}\delta\mathbf{u}_k^\mathrm{T}\mathbf{R}_k\delta\mathbf{u}_k \\ &+ \frac{1}{2}\delta\mathbf{u}_k^\mathrm{T}\mathbf{N}_k\delta\mathbf{x}_k + \frac{1}{2}\delta\mathbf{x}^\mathrm{T}\mathbf{M}_k\delta\mathbf{u}_k,\end{aligned}$$

where the terms $q_k, \mathbf{q}_k, \mathbf{r}_k, \mathbf{Q}_k, \mathbf{R}_k, \mathbf{N}_k$ and $\mathbf{M}_k$ are defined as follows

$$\begin{aligned} q_k &\equiv \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t, \\ \mathbf{q}_k &\equiv \mathcal{L}_\mathbf{x}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t, \\ \mathbf{r}_k &\equiv \mathcal{L}_\mathbf{u}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t \\ \mathbf{Q}_k &\equiv \ \mathcal{L}_{\mathbf{xx}}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t, \\ \mathbf{R}_k &\equiv \mathcal{L}_{\mathbf{uu}}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t, \\ \mathbf{N}_k &\equiv \mathcal{L}_{\mathbf{xu}}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t, \\ \mathbf{M}_k &\equiv \mathcal{L}_{\mathbf{xu}}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t. \end{aligned} \tag{12}$$

Subscripts of $\mathcal{L}(\cdot)$ indicate derivatives, which can be computed analytically. We will use this subscript rule for $V(\cdot)$ as well. The detailed formulations are not given due to space limit. In addition the term $V(\mathbf{x}_{k+1}, k+1)$ is also expanded

$$\begin{aligned} &V(\mathbf{x}_{k+1}, k+1) = V(\bar{\mathbf{x}}_{k+1} + \delta\mathbf{x}_{k+1}, k+1) \\ &= V(\bar{\mathbf{x}}_{k+1}, k+1) + V_{\mathbf{x}_{k+1}}^\mathrm{T}\delta\mathbf{x}_{k+1} + \frac{1}{2}\delta\mathbf{x}_{k+1}^\mathrm{T}V_{\mathbf{xx}_{k+1}}\delta\mathbf{x}_{k+1}. \end{aligned} \tag{13}$$

By taking the expectation we have:

$$\begin{aligned} &\mathbb{E}\Big[V(\bar{\mathbf{x}}_{k+1} + \delta\mathbf{x}_{k+1}, k+1)\Big] = V(\bar{\mathbf{x}}_{k+1}, k+1) \\ &+ V_{\mathbf{x}_{k+1}}^\mathrm{T}\mathbf{A}_k\delta\mathbf{x}_k + V_{\mathbf{x}_{k+1}}^\mathrm{T}\mathbf{B}_k\delta\mathbf{u}_k \\ &+ \frac{1}{2}\delta\mathbf{u}_k^\mathrm{T}\mathbf{B}_k^\mathrm{T}V_{\mathbf{xx}_{k+1}}\mathbf{B}_k\delta\mathbf{u}_k + \frac{1}{2}\delta\mathbf{x}_k^\mathrm{T}\mathbf{A}_k^\mathrm{T}V_{\mathbf{xx}_{k+1}}\mathbf{A}_k\delta\mathbf{x}_k \\ &+ \frac{1}{2}\delta\mathbf{u}_k^\mathrm{T}\mathbf{B}_k^\mathrm{T}V_{\mathbf{xx}_{k+1}}\mathbf{A}_k\delta\mathbf{x}_k + \frac{1}{2}\delta\mathbf{x}_k^\mathrm{T}\mathbf{A}_k^\mathrm{T}V_{\mathbf{xx}_{k+1}}\mathbf{B}_k\delta\mathbf{u}_k. \end{aligned}$$

where the terms $V_{\mathbf{x}_{k+1}}, V_{\mathbf{xx}_{k+1}}$ are defined as

$$\begin{aligned} V_{\mathbf{x}_{k+1}} &= \nabla_\mathbf{x} V(\mathbf{x}, k+1)\Big|_{\mathbf{x}=\bar{\mathbf{x}}_{k+1}} \\ V_{\mathbf{xx}_{k+1}} &= \nabla_{\mathbf{xx}} V(\mathbf{x}, k+1)\Big|_{\mathbf{x}=\bar{\mathbf{x}}_{k+1}}. \end{aligned}$$

C. Computing the local optimal control

To find the optimal control, we compute the local variations in control $\delta\mathbf{u}^*$ that maximize the value function

$$\begin{aligned}\delta\mathbf{u}_k^* &= \arg\max_{\delta\mathbf{u}}(\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\mathrm{d}t + V(\mathbf{x}_{k+1}, k+1) \\ &= \mathbf{l}_k + \mathbf{L}_k\delta\mathbf{x}_k,\end{aligned} \tag{14}$$

where

$$\mathbf{l}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k, \qquad \mathbf{L}_k = -\mathbf{H}_k^{-1}\mathbf{G}_k. \tag{15}$$

The terms $\mathbf{g}, \mathbf{G}, \mathbf{H}$ are specified as

$$\begin{aligned} \mathbf{g}_k &= \mathbf{r}_k + \mathbf{B}_k^\mathrm{T}V_{\mathbf{x}_{k+1}}, \quad \mathbf{H}_k = \mathbf{R}_k + \mathbf{B}_k^\mathrm{T}V_{\mathbf{xx}_{k+1}}\mathbf{B}_k, \\ \mathbf{G}_k &= \frac{1}{2}\mathbf{N}_k + \frac{1}{2}\mathbf{M}_k + \mathbf{B}_k^\mathrm{T}V_{\mathbf{xx}_{k+1}}\mathbf{A}_k. \end{aligned} \tag{16}$$

To guarantee convergence we would like to ensure positive definiteness of $\mathbf{H}_k$, we will discuss how to achieve that in the next section. The local optimal control $\mathbf{u}^*$ is computed as

$$\mathbf{u}_k^* = \bar{\mathbf{u}}_k + \alpha\delta\mathbf{u}_k^*, \tag{17}$$

where α is a weighting factor. By plugging the optimal control back into the value function, splitting terms into zero, first and second order in $\delta\mathbf{x}$ we have:

$$V(\mathbf{x} + \delta\mathbf{x}, t) = V_k + V_{\mathbf{x}_k}^\mathrm{T}\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^\mathrm{T}V_{\mathbf{xx}_k}\delta\mathbf{x}, \tag{18}$$

where V_k, $V_{\mathbf{x}_k}$, $V_{\mathbf{xx}_k}$ can be computed backward in time

$$\begin{aligned} V_k &= V_{k+1} + q_k + \mathbf{g}_k^\mathrm{T}\mathbf{l}_k + \frac{1}{2}\mathbf{l}_k^\mathrm{T}\mathbf{H}_k\mathbf{l}_k, \\ V_{\mathbf{x}_k} &= \mathbf{q}_k + \mathbf{L}_k^\mathrm{T}\mathbf{g}_k + \mathbf{L}_k^T\mathbf{H}_k\mathbf{l}_k + \mathbf{G}_k^\mathrm{T}\mathbf{l}_k + \mathbf{A}_k^\mathrm{T}V_{\mathbf{x}_{k+1}}, \\ V_{\mathbf{xx}_k} &= \mathbf{Q}_k + \mathbf{L}_k^T\mathbf{H}_k + \mathbf{L}_k^T\Big(\mathbf{N}_k + \mathbf{B}_k^\mathrm{T}V_{\mathbf{xx}_{k+1}}\mathbf{A}_k\Big) \\ &+ \Big(\mathbf{M}_k + \mathbf{A}_k^\mathrm{T}V_{\mathbf{xx}_{k+1}}\mathbf{B}_k^\mathrm{T}\Big)\mathbf{L}_k + \mathbf{A}_k^\mathrm{T}V_{\mathbf{xx}_{k+1}}\mathbf{A}_k. \end{aligned} \tag{19}$$

Therefore, the second order approximation of the value function will be propagated backward in time iteratively. The optimal control sequence will be applied to generate a local optimal state trajectory forward in time. Next, we will discuss two algorithms based on the GPDDP framework.

V. SUMMARY OF ALGORITHMS

In this section, we present two algorithms based on the GPDDP framework. The proposed algorithms are summarized in **Algorithm 1**(online version) and **Algorithm 2** (off-line version) and are designed for different purposes. The online algorithm requires interaction with the physical system at every iteration. In particular an control policy is obtained at every iteration, and it is applied to the physical system to generate a trajectory to update the GP model. Therefore the online version is suitable for applications in reinforcement learning. In the off-line version, the GP model of the system dynamics is learned based on state and control trajectories collected initially. Then the GPDDP is applied to find the optimal control without interaction with the physical system. The offline algorithm collects training data as initialization and applies the optimal control policy to the real physical system after convergence. Therefore, the offline version of GPDDP is suitable for tasks such as autonomous flight trajectory optimization where repetitive forward sampling from the dynamics is unrealistic. Both algorithms contain the following tasks:

Dynamics model learning: Given a nominal trajectory in states and controls, the nonlinear dynamics can be approximated locally by the linearized GP model as introduced in Section IV-A. The approximated model is Bayesian nonparametric and takes into account uncertainties/stochaticities of the training data. In this paper, we use trajectory sampling to generate the training data that consist of pairs, e.g., $(\{\mathbf{x}_k, \mathbf{u}_k\}, \{\mathbf{x}_{k+1} - \mathbf{x}_k\})$. For the case of generating multiple trajectories based on the same nominal trajectory, we use small variations of $\bar{\mathbf{u}}$.

Back-propagation: Compute the local optimal control law $\delta\mathbf{u}_k^*$ as in (14) and cost to go function $V(\mathbf{x} + \delta\mathbf{x}, t_k)$ as in (18) in a backward pass through time. The details about this computation have been shown in Section IV-C. In order to improve numerical stability we would like to ensure positive definiteness of the Hessian matrix $\mathbf{H}_k$. In both algorithm we employ the Levenberg-Marquardt related trick (also used in [2]). When $\mathbf{H}_k$ has negative eigenvalues, first we compute the eigenvalue decomposition, i.e., $[\mathbf{V}, \mathbf{D}]$ =eig($\mathbf{H}_k$) and replace all negative elements of $\mathbf{D}$ with 0. Then a small positive value λ is added to $\mathbf{D}$. The modified matrix is obtained as $\tilde{\mathbf{H}}_k = \mathbf{V}\mathbf{D}\mathbf{V}^{\mathrm{T}}$. Its inverse is computed as $\tilde{\mathbf{H}}_k^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^{\mathrm{T}}$.

Forward propagation: Apply the local optimal control obtained from back-propagation to the dynamics (either the stochastic dynamics or the GP approximation). This step will give us a locally optimal trajectory, which will be used as the new nominal trajectory to linearize the dynamics in the next iteration. In both algorithm we implement line search by adding a parameter $\varepsilon > 0$ such that $\delta\mathbf{u}_k^* = \varepsilon\mathbf{l}_k + \mathbf{L}_k\delta\mathbf{x}_k$. Initially $\varepsilon = 1$, when the trajectory generated by the learned policy has a higher cost than the current one, the policy would be rejected and decrease ε. Whenever the policy is accepted we reset $\varepsilon = 1$. This method has also been used in [2] to encourage convergence.

Given: Nominal trajectory
Goal : Optimized trajectory

for $i = 1$ **to** I_{max} **do**
  Generate trajectories by sampling from the dynamics (1) and forward propagation ;
  Obtain training pairs, e.g., $(\{\mathbf{x}_k, \mathbf{u}_k\}, \{\mathbf{x}_{k+1} - \mathbf{x}_k\})$;
  Linearize the GP dynamics around $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ according to Section IV-A;
  Backpropagate to get the local optimal control $\delta\mathbf{u}^*$ and value function $V(\mathbf{x} + \delta\mathbf{x}, t)$ according to (14) (18);
  Forward propagate (1) by applying the optimal control $\mathbf{u}_k^* = \bar{\mathbf{u}}_k + \alpha\delta\mathbf{u}_k^*$, obtain a new trajectory $(\mathbf{x}^*, \mathbf{u}^*)$;
  if *Converge* **then** Break the **for** loop;
  Set $\bar{\mathbf{x}} = \mathbf{x}^*$ and $\bar{\mathbf{u}} = \mathbf{u}^*$;
end
Apply the optimal control $\mathbf{u}^*$ to (1), obtain the optimized trajectory.

Algorithm 1: On-line GPDDP

Given: Nominal trajectory
Goal : Optimized trajectory

Generate trajectories by sampling from the dynamics (1) and forward propagation;
Obtain training pairs, e.g., $(\{\mathbf{x}_k, \mathbf{u}_k\}, \{\mathbf{x}_{k+1} - \mathbf{x}_k\})$;
for $i = 1$ **to** I_{max} **do**
  linearize the GP dynamics around $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ according to Section IV-A;
  Backpropagate to obtain the local optimal control $\delta\mathbf{u}^*$ and value function $V(\mathbf{x} + \delta\mathbf{x}, t_k)$ according to (14) (18);
  Forward propagate the GP dynamics by applying the optimal control $\mathbf{u}_k^* = \bar{\mathbf{u}}_k + \alpha\delta\mathbf{u}_k^*$, obtain a new trajectory $(\mathbf{x}^*, \mathbf{u}^*)$;
  if *Converge* **then** Break the **for** loop;
  Set $\bar{\mathbf{x}} = \mathbf{x}^*$ and $\bar{\mathbf{u}} = \mathbf{u}^*$;
end
Apply the optimal control $\mathbf{u}^*$ to (1), obtain the optimized trajectory.

Algorithm 2: Off-line GPDDP

A. Computational complexity

For DDP, the heaviest computational burden comes from linearization of the given dynamics up to the second order. Generally the analytic forms of partial derivatives involved in linearization are not available, which are usually approximated using finite differences. For GPDDP, we compute all Jacobian matrices analytically. For systems with high dimensional state spaces, this linearization scheme is much more efficient than finite differences, see Fig. 4 for example. For GPDDP, the major part of computational effort is devoted to GP inferences. In particular, the complexity of computing one-step moment matching is $\mathcal{O}(N^2n^2(n+m))$ and the

inverse of kernel matrix $\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}})^{-1}$ is $\mathcal{O}(N^3)$ where N is the number of the training data. The computational efficiency can be improved in various ways. For the online case, instead of generating new training data at each iteration, we may use sparse GP [20] to add/remove data to reduce computational burden. For the offline case, local GP model [13] may be employed to compute local kernel matrices with smaller size, and approximate the predictive mean of GP by the weighted mean of each local model.

VI. Numerical Examples

In this section, we provide three numerical examples to evaluate the proposed GPDDP framework.

A. Inverted pendulum

The dynamics of inverted pendulum can be described as follow:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{x}_2 \\ \frac{1}{I}\Big(mgl\sin(\mathbf{x}_1) - b\mathbf{x}_2 + \mathbf{u}\Big) \end{bmatrix}, \qquad (20)$$

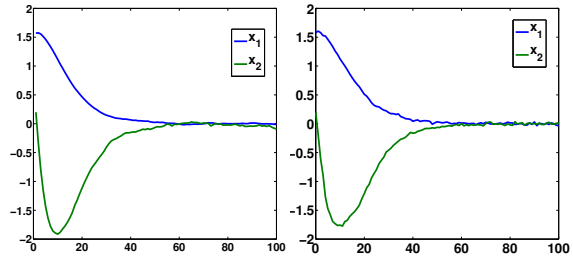
and $\mathbf{g} = [0.5; 1]$. Where $\mathbf{x}_1$ is the position and $\mathbf{x}_2$ is the velocity. The task is to bring the pendulum from initial position $[\frac{\pi}{2}\ \ 0]$ to the inverted position with zero velocity $[0\ \ 0]$. Both online and offline GP-DDP algorithms were applied to the system. The results are shown in Fig. 2. Both algorithms are able to complete the task and the finally optimized trajectories are similar. It should be noticed that the offline algorithm requires more iterations to converge than the online one, since the online scheme features more accurate approximation of the dynamics by sampling from (20) at every iteration. By doing so, the online algorithm also cost more computational effort at every iteration than the offline scheme.

B. Six-link robotic arm

The six-link arm model consist of six links of equal length and mass, connected in an open chain with resolute joints. The system has 6 degrees of freedom, and 12 state dimensions (angle and angular velocity for each joint). The goal for the first 3 joints is to move to $\frac{\pi}{4}$ and the rest 3 joints to $-\frac{\pi}{4}$. The desired velocities for all 6 joints are zeros. We apply the online GPDDP on this task and sample 1 trajectory from the true dynamics at each iteration. For comparison purpose, we also apply the standard DDP to this system with a given dynamics model. We perform all linearizations in DDP using finite differences. The optimized state trajectories are shown in Fig. 3 and a comparison is made in Fig. 4. It can be seen that GPDDP (with learned dynamics) generates similar trajectories as DDP does (with known dynamics). As we mentioned in section V-A, GPDDP linearize the learned GP model by computing Jacobian matrices analytically, which is significantly more efficient than finite-differences applied in DDP. GPDDP requires slightly more iterations to converge than DDP to obtain the optimized trajectory since the learned model is updated during optimization.

C. Helicopter hovering

The task of autonomous helicopter hovering is challenging because the helicopter systems are generally dynamically unstable. DDP has been used for autonomous helicopter tasks such as hovering and achieved state-of-the-art performances [10]. As most DDP applications, an accurate dynamical model of the helicopter system is essential. [10] assumes a parametrization of the helicopter dynamics and used flight data to estimate those parameters. The proposed GPDDP does not assume any parametrization of the dynamics and relies only on data. We applied the proposed GPDDP framework to the helicopter hovering task. The goal is to stabilize the helicopter from initial disturbances. Since sampling from the physical system online is not realistic, most GP-based reinforcement learning approaches (e.g.,[16]) are not directly applicable in this case, we use the offline scheme for this task. As in [10], the states can be described by positions $(\mathbf{x}, \mathbf{y}, \mathbf{z})$, velocities $(\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z)$ and angular velocities $(\mathbf{w}_x, \mathbf{w}_y, \mathbf{w}_z)$. Control inputs are aileron, elevator, rudder, and collective pitch control, which are denoted by $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$ and $\mathbf{u}_4$ respectively. The desired steady-states are all zeros. In our simulation, we generate trajectories from the parametric model (with stochastic terms) learned in [10] . Results are shown in Fig. 5. Notice that $\mathbf{u}_1, \mathbf{u}_2$ and $\mathbf{u}_3$ control the orientation of helicopter, $\mathbf{u}_4$ adjust the thrust of the main motor. Positive $\mathbf{u}_4$ force the main motor to blow air downward relative to the helicopter, and negative $\mathbf{u}_4$ blow air upward relative to the helicopter. GPDDP successfully accomplish this task with 10 sampled trajectories.

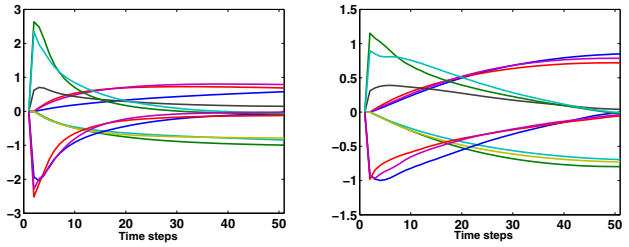


(a) (b)

Fig. 2: Optimized state trajectories for the inverted pendulum task. In all plots, the X-axis is the time steps. (a) Online GPDDP. (b) Offline GPDDP.

VII. Conclusions

In this paper, we propose a data-driven differential dynamic programming framework based on GPs, called GPDDP. The proposed framework is probabilistic model-based approach that is applicable to a general class of systems with unknown dynamics. Because of GP's Bayesian nonparametric nature, GPDDP takes into account model uncertainty and relies only on sampled data without any assumed structure of the system. Building on this framework, we also presented online and offline algorithms for implementations in different scenarios. The framework was evaluated in three numerical examples.



(a) (b)

Fig. 3: Optimized state trajectories for six-link arm task. (a) DDP. (b) GPDDP.

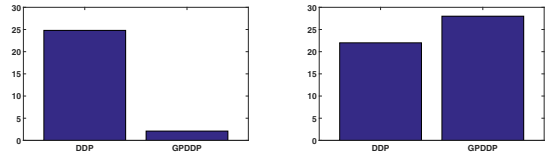
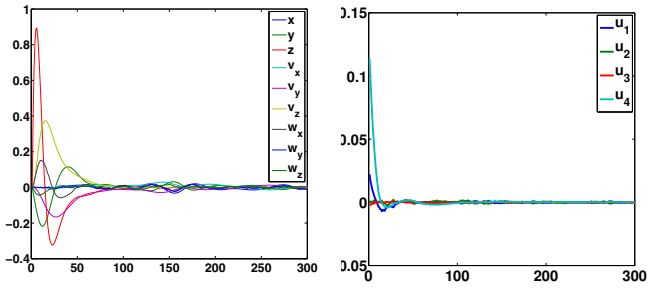


Fig. 4: Comparison of DDP and GPDDP for the six-link arm task. Left figure: time consumed at each iteration for linearization (sec). Right: number of iterations to obtain the optimized trajectory.



(a) (b)

Fig. 5: Simulation results for the helicopter hovering task. In all plots, the X-axis is the time steps. (a) optimized state trajectories. (b) optimal controls.

In general, the family of Bayesian nonparametric trajectory optimization methods combines the attractive characteristics of probabilistic model learning and model-based trajectory optimization. The applicability of the proposed framework to physical robotic and aerospace systems is ongoing work. Future research includes the use of the predictive variances for risk sensitive controls and the improvement of the computational efficiency by using sparse GP, local regression, etc.

REFERENCES

[1] D. Jacobson and D. Mayne. Differential dynamic programming. 1970.

[2] E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference (ACC)*, pages 300–306, 2005.

[3] Y. Tassa, T. Erez, and W. D. Smart. Receding horizon differential dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[4] E. Theodorou, Y. Tassa, and E. Todorov. Stochastic differential dynamic programming. In *American Control Conference (ACC)*, pages 1125–1132. IEEE, 2010.

[5] D. Mitrovic, S. Klanke, and S. Vijayakumar. Adaptive optimal feedback control with learned internal dynamics models. In *From Motor Learning to Interaction Learning in Robots*, pages 65–84. Springer, 2010.

[6] J. Van Den Berg, S. Patil, and R. Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[7] S. Levine and V. Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 207–215. 2013.

[8] Y. Tassa, N. Mansard, and E. Todorov. Control-limited differential dynamic programming. In *International Conference on Robotics and Automation (ICRA)*, 2014.

[9] J. Morimoto, G. Zeglin, and C. G Atkeson. Minimax differential dynamic programming: Application to a biped walking robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1927–1932. IEEE, 2003.

[10] P. Abbeel, A. Coates, M. Quigley, and A. Y Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems (NIPS)*, 19:1, 2007.

[11] C.K.I Williams and C.E. Rasmussen. *Gaussian processes for machine learning*. MIT Press, 2006.

[12] J. Ko and D. Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

[13] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model learning with local gaussian process regression. *Advanced Robotics*, 23(15):2015–2034, 2009.

[14] M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7):1508–1524, 2009.

[15] M. P. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, pages 465–472, 2011.

[16] Y. Pan and E. Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1907–1915, 2014.

[17] M. Deisenroth, D. Fox, and C. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Trans. Pattern Anal. Mach. Intell. (PAMI)*, 37(2):408–423, 2015.

[18] J. Quinonero Candela, A. Girard, J. Larsen, and C. E. Rasmussen. Propagation of uncertainty in bayesian kernel models-application to multiple-step ahead forecasting. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003.

[19] A. Girard, C.E. Rasmussen, J. Quinonero-Candela, and R. Murray-Smith. Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[20] L. Csató and M. Opper. Sparse on-line gaussian processes. *Neural Computation*, 14(3):641–668, 2002.