Efficient Reinforcement Learning via Probabilistic Trajectory Optimization

Yunpeng Pan, George I. Boutselis, and Evangelos A. Theodorou

***Abstract*— We present a trajectory optimization approach to reinforcement learning in continuous state and action spaces, called probabilistic differential dynamic programming (PDDP). Our method represents systems dynamics using Gaussian processes (GPs), and performs local dynamic programming iteratively around a nominal trajectory in Gaussian belief spaces. Different from model-based policy search methods, PDDP does not require a policy parameterization and learns a time-varying control policy via successive forward–backward sweeps. A convergence analysis of the iterative scheme is given, showing that our algorithm converges to a stationary point globally under certain conditions. We show that prior model knowledge can be incorporated into the proposed framework to speed up learning, and a generalized optimization criterion based on the predicted cost distribution can be employed to enable risk-sensitive learning. We demonstrate the effectiveness and efficiency of the proposed algorithm using nontrivial tasks. Compared with a state-of-the-art GP-based policy search method, PDDP offers a superior combination of learning speed, data efficiency, and applicability.**

***Index Terms*— Dynamic programming, Gaussian processes (GPs), optimal control, reinforcement learning (RL), trajectory optimization.**

I. Introduction

REINFORCEMENT learning (RL) for continuous state-action spaces promises control of robotic and autonomous systems with many degrees of freedom and unknown dynamics. While model-free RL methods have demonstrated promising results in learning control applications [1]–[4], they typically require human expert demonstrations and a relatively large number direct interactions with the physical system. In contrast, model-based RL was developed to address the issue of sample inefficiency by learning transition dynamics models explicitly from data, which can also help to provide better generalization [5], [6]. However, model-based methods suffer from two severe issues: 1) classical value function approximation methods [7], [8] and modern global policy search methods [6] are computationally inefficient for moderate to high-dimensional problems and 2) model errors significantly degrade the performance. To address the aforementioned issues, in this paper, we propose an RL framework that relies on differential dynamic programming (DDP) and Gaussian process (GP) regression.

Originally introduced in the 70s [9], DDP solves nonlinear optimal control problems via successive local approximations of dynamics and cost functions along nominal trajectories. DDP iteratively generates locally optimal feedforward and feedback control policies along with an optimal state trajectory. Compared with global optimal control approaches, DDP shows superior computational efficiency and scalability to high-dimensional problems. In the last decade, variations of DDP have been proposed within the control, robotics, and machine learning communities. These variations include generalizations to stochastic systems [10], [11], extensions to model predictive control [12], and min–max and cooperative game theoretic formulations [13], [14]. However, DDP relies on explicit dynamics models and it is sensitive to model errors. To address these issues, various data-driven approaches have been developed, such as minimax DDP using receptive field weighted regression (RFWR) [13] and DDP using expert-demonstrated trajectories [15]. One issue with these approaches is that the learned models are deterministic and do not explicitly take into account any model uncertainty. As a consequence, the aforementioned methods have either limited model expressiveness due to the fixed model structure [15], or require a large amount of data to learn a good transition model [13], [16]. In this paper, we perform modeling and prediction using GPs.

GPs are used to define distributions over continuous functions and offer a powerful way to perform Bayesian nonparametric estimation of functions, e.g., unknown transition dynamics. Over the last decade, there has been an increasing interest in developing control/RL algorithms using GPs. For instance, the work by Rasmussen and Kuss [17] is one of the first GP-based RL algorithm; Nguyen-Tuong *et al.* [18]–[20] explored inverse dynamics learning and tracking control via local GPs and semiparametric GPs; Deisenroth and Rasmussen [21] and Deisenroth *et al.* [22] proposed model-based policy search using GPs; Hemakumara and Sukkarieh [23] used GPs for unmanned aerial vehicle controls and Chowdhary *et al.* [24] incorporated GPs for adaptive controls. Dallaire *et al.* [25]

Manuscript received February 23, 2016; revised January 30, 2017 and August 22, 2017; accepted October 3, 2017. This work was supported in part by ARO under Grant W911NF-16-1-0390, in part by NSF-NRI under Grant 1426945, and in part by the Institute for Robotics and Intelligent Machines at Georgia Tech. The work of G. I. Boutselis was supported by the A. S. Onassis Foundation. *(Corresponding author: Yunpeng Pan.)*

Y. Pan is with the JD-X Silicon Valley Research Center, JD.com American Technologies Corporation, Santa Clara, CA 95054 USA (e-mail: yunpeng.pan@jd.com).

G. I. Boutselis and E. A. Theodorou are with the Department of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: gbouts@gatech.edu; evangelos.theodorou@gatech.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2017.2764499

use GPs to learn the transition, observation, and reward models in POMDPs. These works have demonstrated the remarkable applicability of GP-based methods in robotics and autonomous learning.

In this paper, we integrate GPs into trajectory optimization and propose a probabilistic variant of DDP, called probabilistic DDP (PDDP). The resulting algorithm performs probabilistic trajectory optimization that combines the benefits of DDP and GP inference. The major characteristics of PDDP are summarized as follows.

1) It features data efficiency that is comparable to the state-of-the-art method.
2) Computationally, it is significantly faster than other GP-based policy search methods.
3) Convergence is guaranteed under certain conditions. Our theoretical analysis generalizes previous work.
4) Prior model knowledge and risk-sensitive criterion can be incorporated into our proposed framework.

PDDP is related to a number of recently developed model-based RL approaches that use GPs to represent dynamics models. In particular, the PILCO framework developed by Deisenroth *et al.* [22] has achieved unprecedented performances in terms of data efficiency. PILCO requires an external optimizer (e.g., CG or BFGS) for solving nonconvex optimization to obtain optimal policy parameters. In contrast, PDDP does not require a policy parameterization nor an extra optimizer.[1] Compared with other DDP-related approaches for stochastic [10], [11], unknown [16], [26], or partially known dynamics [13], PDDP explicitly takes into account model uncertainty and features the attractive characteristics of GP-based methods, i.e., data efficiency. In addition, we incorporate cost uncertainty into the optimization criterion for risk-sensitive learning. We will further discuss the similarities and differences between our method and the aforementioned methods in Section IV-H.

The rest of this paper is structured into the following sections. Section II provides the problem formulation, important definitions, and the notation considered in this paper. In Section III, we address the theoretical foundations of modeling and prediction using GPs for unknown or partially known dynamical systems. Section IV derives and analyzes the PDDP framework, which is the main algorithmic development of this paper. In Section V, we provide applications of the proposed PDDP framework using nontrivial RL tasks. Section VI discusses the limitations and possible extensions of the proposed work, and finally concludes this paper.

II. Preliminaries

In this section, we define the general formulation of model-based RL, GPs, and other important technical terms used in this paper.

A. Problem Formulation

We consider a dynamical system described by the following differential equation:

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

[1]We use an off-the-shelf optimizer for kernel parameter optimization but not policy optimization.

where $\mathbf{x} \in \mathbb{R}^n$ is the state, $\mathbf{u} \in \mathbb{R}^m$ is the control, and $\omega \in \mathbb{R}^p$ is the standard Brownian motion noise. $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and $\mathbf{C} \in \mathbb{R}^{n \times p}$ are the transition dynamics function and diffusion matrix. The finite-horizon control problem is defined as finding a sequence of state and a control policy π that minimize

$$J(\mathbf{x}(t_0)) = \mathbb{E}_{\mathbf{x}}\left[h(\mathbf{x}(T)) + \int_{t_0}^{T} \mathcal{L}(\mathbf{x}(t), \pi(\mathbf{x}(t), t))dt\right] \tag{2}$$

where $h : \mathbb{R}^n \to \mathbb{R}$ is the terminal cost function and $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the instantaneous cost rate. The control policy $\mathbf{u}(t) = \pi(\mathbf{x}(t), t)$ is a function that maps states and time to actions. The cost $J(\mathbf{x}(t_0))$ is defined as the expectation of the total cost accumulated from t_0 to T. $\mathbb{E}_{\mathbf{x}}$ denotes the expectation operator with respect to $\mathbf{x}$. We assume that the states are fully observable. For the rest of our analysis, we discretize the time using the Euler scheme as $k = 1, 2, \ldots, H$ with time step $\Delta t = ((T - t_0)/(H - 1))$. In order to simplify notation, we use subscripts to denote time steps for time-varying variables, e.g., $\mathbf{x}_k = \mathbf{x}(t_k)$. The discretized system dynamics can be written as $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k + \mathbf{C}\sqrt{\Delta t}\xi_k$, where $\Delta\mathbf{x}_k = \Delta t f(\mathbf{x}_k, \mathbf{u}_k)$ and ξ_k is i.i.d $\mathcal{N}(0, \mathbf{I})$. We also define $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \Delta t f(\mathbf{x}_k, \mathbf{u}_k)$ for simplicity.

The goal of optimal control and RL is to find the control policy $\pi(\mathbf{x}(t), t)$ that minimizes the cost [see (2)]. Finding the globally optimal control policy is computationally intractable except for linear or low-dimensional systems. In this paper, we seek locally optimal policy, which is an approximation of the globally optimal policy in the neighborhood of a nominal trajectory.

B. Important Definitions

In the following, we briefly introduce the definitions of *GP* and *belief*, which are the core concepts used in this paper.

Definition 1: A *GP* is a collection of random variables, any finite subset of which has a joint Gaussian distribution [27].

Alternatively, a GP is defined as a distribution over function, or a generalization of the Gaussian distribution to an infinite-dimensional function space. Similar to a Gaussian distribution, a GP is completely specified by a mean function and a covariance function, i.e., $p(\mathbf{f}(\mathbf{x})) = \mathcal{GP}(\mathbf{m}(\mathbf{x}), \mathbf{k}(\mathbf{x}, \mathbf{x}'))$, where

$$\mathbf{m}(\mathbf{x}) = \mathbb{E}_{\mathbf{f}}[\mathbf{f}(\mathbf{x})], \quad \mathbf{k}(\mathbf{x}, \mathbf{x}') = \mathbb{COV}_{\mathbf{f}}[\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x}')] \tag{3}$$

where $\mathbb{E}_{\mathbf{f}}$ and $\mathbb{COV}_{\mathbf{f}}$ denote the expectation and covariance operators with respect to the random function $\mathbf{f}$. The covariance function $\mathbf{k}(\mathbf{x}, \mathbf{x}')$ is also called a *kernel*.

Definition 2: A belief $\mathbf{b}$ of a dynamical system is defined as the probability distribution of the state $\mathbf{x}$ given a set of sampled control inputs and state observations.

In this paper, we assume that the state is observable, and that a belief can be represented by a Gaussian distribution over the state of a dynamical system, i.e., $p(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The belief can be obtained via approximate inference in probabilistic models given data. In Section III, we discuss modeling, inference, and learning based on GPs in further detail.

III. Probabilistic Model Learning and Inference

One of the major assumptions of deterministic model-based RL methods is that the optimal policy obtained using the learned model is close to the nominal optimal policy, which could be obtained if the true model is available. Due to this assumption, the performance of most deterministic model-based RL methods degrades when the learned model is far from the actual dynamics. Probabilistic models on the other hand can explicitly incorporate uncertainty when predicting the system behaviors and reduce the effect of model errors. In our approach, we use a probabilistic model learning and approximate inference scheme based on GPs.

A. Learning and Inference Based on Gaussian Processes

Learning a functional mapping from state-control pair $\tilde{\mathbf{x}} = (\mathbf{x}, \mathbf{u}) \in \mathbb{R}^{n+m}$ to state transition $\Delta\mathbf{x}$ can be viewed as an probabilistic inference with the goal of inferring $\Delta\mathbf{x}$ given $\tilde{\mathbf{x}}$. In order to perform inference about the transition dynamics, we need a prior belief about the latent function. Without any prior knowledge of the model, we may assume a prior mean function $\mathbf{m}(\cdot) = 0$ and a squared exponential covariance function plus a noise covariance

$$\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{W}^{-1}(\mathbf{x}_i - \mathbf{x}_j)\right) \quad (4)$$

and $\mathbb{COV}(\Delta\mathbf{x}_i, \Delta\mathbf{x}_j) = \mathbf{k}(\mathbf{x}_j, \mathbf{x}_j) + \sigma_\omega^2 \delta_{ij}$, where δ_{ij} is a Kronecker delta which is one if and only if $i = j$ and zero otherwise. $\mathbf{W} = \text{diag}([\, l_1^2 \ \ldots \ l_{n+m}^2 \,])$. The hyperparameters $\boldsymbol{\theta}$ of the kernel consist of the signal variance σ_f^2 that controls the variations of the function values from its mean, the noise variance σ_ω^2 that determines the noise magnitude we have in the data, and the characteristic length scales for input space $l_1, \ldots, l_{n+m}$ that describe the smoothness of the function. The kernel function is interpreted as a similarity measure of random variables. In contrast to parametric approaches that rely on assumed structures and a finite number of parameters, the GP approach puts a prior on function directly; therefore, it is nonparametric.

1) Prediction via GP Regression: Given a collection of sampled state-control pairs $\tilde{\mathbf{X}} = \{(\mathbf{x}_1^s, \mathbf{u}_1^s), \ldots \ (\mathbf{x}_N^s, \mathbf{u}_N^s)\}$, and the corresponding state transition $\Delta\mathbf{X} = \{\Delta\mathbf{x}_1^s, \ldots, \Delta\mathbf{x}_N^s\}$, the joint distribution of the observed output and the output corresponding to a given test state-control pair $\tilde{\mathbf{x}}^* = (\mathbf{x}^*, \mathbf{u}^*)$ can be written as

$$p\begin{pmatrix}\Delta\mathbf{X} \\ \mathbf{f}(\tilde{\mathbf{x}}^*)\end{pmatrix} = \mathcal{N}\left(0, \begin{bmatrix}\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_\omega^2 \mathbf{I} & \mathbf{k}(\tilde{\mathbf{X}}, \tilde{\mathbf{x}}^*) \\ \mathbf{k}(\tilde{\mathbf{x}}^*, \tilde{\mathbf{X}}) & \mathbf{k}(\tilde{\mathbf{x}}^*, \tilde{\mathbf{x}}^*)\end{bmatrix}\right)$$

where $\mathbf{K}$ is a matrix with entries $\mathbf{K}_{ij} = \mathbf{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$. The posterior distribution, or predictive distribution, can be obtained by conditioning the joint distribution on the observed state transitions. Assuming independent outputs (no correlation between each output dimension) and given a test state-action pair $\tilde{\mathbf{x}}_k = (\mathbf{x}_k, \mathbf{u}_k)$ at time step k, the one-step predictive distribution is $p(\mathbf{f}\ (\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}}, \Delta\mathbf{X}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k})$, where the mean and variance are specified as

$$\begin{aligned}\boldsymbol{\mu}_{f_k} &= \mathbf{k}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}})(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_w \mathbf{I})^{-1} \Delta\mathbf{X} \\ \boldsymbol{\Sigma}_{f_k} &= \mathbf{k}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_k) - \mathbf{k}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}})(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_w \mathbf{I})^{-1} \mathbf{k}(\tilde{\mathbf{X}}, \tilde{\mathbf{x}}_k). \quad (5)\end{aligned}$$

The prediction of successor state $\mathbf{x}_{k+1}$ is Gaussian distributed $p(\mathbf{x}_{k+1}) = \mathcal{N}(\boldsymbol{\mu}_{k+1}, \boldsymbol{\Sigma}_{k+1})$, where the state mean and variance are $\boldsymbol{\mu}_{k+1} = \mathbf{x}_k + \boldsymbol{\mu}_{f_k}$ and $\boldsymbol{\Sigma}_{k+1} = \boldsymbol{\Sigma}_{f_k}$. When performing two-step prediction, the input state-control pair $\tilde{\mathbf{x}}_{k+1}$ becomes uncertain. Here, we define the input distribution over state-control pair at k as $p(\tilde{\mathbf{x}}_k) = p(\mathbf{x}_k, \mathbf{u}_k) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)$. Thus, the distribution over state transition becomes $p(\mathbf{f}(\tilde{\mathbf{x}}_k)) = \int p(\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k) p(\tilde{\mathbf{x}}_k) d\tilde{\mathbf{x}}_k$. Generally, this predictive distribution cannot be computed analytically and the nonlinear mapping of an input Gaussian distribution leads to a non-Gaussian predictive distribution. However, the predictive distribution can be approximated by a Gaussian via computing the exact posterior mean and variance $p(\mathbf{f}(\tilde{\mathbf{x}}_k)) = \mathcal{N}(\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k})$ [28], [29]. Thus, the state distribution at $k+1$ is also a Gaussian $\mathcal{N}(\boldsymbol{\mu}_{k+1}, \boldsymbol{\Sigma}_{k+1})$ [22]

$$\begin{aligned}\boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{\mu}_{f_k} \\ \boldsymbol{\Sigma}_{k+1} &= \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_{f_k} + \boldsymbol{\Sigma}_{\mathbf{f}_k, \mathbf{x}_k} + \boldsymbol{\Sigma}_{\mathbf{x}_k, \mathbf{f}_k} \quad (6)\end{aligned}$$

where $\boldsymbol{\Sigma}_{\mathbf{f}_k, \mathbf{x}_k}$ is the cross covariance between state and the corresponding state transition. The computation of $\boldsymbol{\mu}_{f_k}$, $\boldsymbol{\Sigma}_{f_k}$, and $\boldsymbol{\Sigma}_{\mathbf{f}_k, \mathbf{x}_k}$ will be discussed in Section III-A2.

2) Approximate Inference via Exact Moment Matching: Given an input joint distribution $\mathcal{N}(\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)$, we employ the moment matching approach [22], [28], [29] to compute the predictive distribution. This technique is also known as assumed density filtering that minimizes the Kullback–Leibler divergence between the true posterior $p(\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)$ and the Gaussian approximation $p(\mathbf{f}(\tilde{\mathbf{x}}_k)) \sim \mathcal{N}(\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k})$ with respect to the natural parameters of the Gaussian $p(\mathbf{f}(\tilde{\mathbf{x}}_k))$, that is

$$\min_{\zeta} \ \text{KL}(p(\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k) \parallel p(\mathbf{f}(\tilde{\mathbf{x}}_k)))$$

with natural parameters $\zeta = (\boldsymbol{\Sigma}_{f_k}^{-1} \boldsymbol{\mu}_{f_k}, (1/2)\boldsymbol{\Sigma}_{f_k}^{-1})$. Next, we compute the moments $\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k}$ in closed form. Applying the law of iterated expectation, the predictive mean $\boldsymbol{\mu}_{f_k}$ is evaluated as

$$\begin{aligned}\boldsymbol{\mu}_{f_k} &= \mathbb{E}_{\mathbf{f}, \tilde{\mathbf{x}}_k}[\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] = \mathbb{E}_{\tilde{\mathbf{x}}_k}[\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k]|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] \\ &= \int \mathbb{E}_{\mathbf{f}}[\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k] \mathcal{N}(\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k) d\tilde{\mathbf{x}}_k.\end{aligned}$$

To make analysis concise, we omit the conditioning on the training pairs $\tilde{\mathbf{X}}$ and $\Delta\mathbf{X}$ and hyperparameters $\boldsymbol{\theta}$. The predictive distributions are explicitly conditioned on the input arguments, i.e., deterministic $\tilde{\mathbf{x}}_k$ or uncertain $\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k$. Next, we compute the predictive covariance matrix, as shown at the bottom of this page, where the variance term on the

$$\boldsymbol{\Sigma}_{f_k} = \begin{bmatrix}\mathbb{VAR}_{\mathbf{f}, \tilde{\mathbf{x}}_k}[\mathbf{f}_1(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] & \ldots & \mathbb{COV}_{\mathbf{f}, \tilde{\mathbf{x}}_k}[\mathbf{f}_1(\tilde{\mathbf{x}}_k), \mathbf{f}_n(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] \\ \vdots & \ddots & \vdots \\ \mathbb{COV}_{\mathbf{f}, \tilde{\mathbf{x}}_k}[\mathbf{f}_n(\tilde{\mathbf{x}}_k), \mathbf{f}_1(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] & \ldots & \mathbb{VAR}_{\mathbf{f}, \tilde{\mathbf{x}}_k}[\mathbf{f}_n(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k]\end{bmatrix}$$

diagonal for output dimension i is obtained using the law of total variance

$$\begin{aligned}&\mathbb{VAR}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\mathbf{f}_i(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k]\\&\quad= \mathbb{E}_{\tilde{\mathbf{x}}_k}[\mathbb{VAR}_{\mathbf{f}}[\mathbf{f}_i(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k]|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k]\\&\qquad+ \mathbb{E}_{\tilde{\mathbf{x}}_k}[\mathbb{E}_{\mathbf{f}}[\mathbf{f}_i(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k]^2|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] - \boldsymbol{\mu}_{f i_k}^2\end{aligned}$$

and the off-diagonal covariance term for output dimension i, j is given by the expression

$$\begin{aligned}&\mathbb{COV}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\mathbf{f}_i(\tilde{\mathbf{x}}_k), \mathbf{f}_j(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k]\\&\quad= \mathbb{E}_{\tilde{\mathbf{x}}_k}[\mathbb{E}_{\mathbf{f}}[\mathbf{f}_i(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k]\mathbb{E}_{\mathbf{f}}[\mathbf{f}_j(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k]|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] - \boldsymbol{\mu}_{f i_k}\boldsymbol{\mu}_{f j_k}.\end{aligned}$$

The input–output cross covariance is formulated as

$$\begin{aligned}\boldsymbol{\Sigma}_{\mathbf{f}_k,\tilde{\mathbf{x}}_k} &= \mathbb{COV}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\tilde{\mathbf{x}}_k, \mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k]\\&= \mathbb{E}_{\tilde{\mathbf{x}}_k}[\tilde{\mathbf{x}}_k\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k]^{\mathsf{T}}|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] - \tilde{\boldsymbol{\mu}}_k\boldsymbol{\mu}_{f_k}^{\mathsf{T}}. \quad (7)\end{aligned}$$

$\boldsymbol{\Sigma}_{\mathbf{f}_k,\mathbf{x}_k} = \mathbb{COV}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\mathbf{x}_k, \mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k]$ can be easily obtained as a submatrix of the above matrix. See [22] and [30] for detailed derivations of the moment matching approach. This technique has been used to perform prediction under uncertain inputs in GP-based estimation [31] and control [32] tasks.

Despite providing the analytic expressions of the moments, the moment matching approach does not provide an explicit error between our Gaussian approximation and the true posterior distribution. An explicit error bound may be achieved by using numerical approximation methods. For instance [33] uses numerical quadrature, and the error bound grows exponentially with the predictive horizon. However, such methods are known to suffer from the *curse of dimensionality*. Hence, the analytic moment-based approach presented above is more suitable for the trajectory optimization framework developed in this paper.

3) Hyperparameters: The hyperparameters $\boldsymbol{\theta}$ are learned by maximizing the log-marginal likelihood of the training outputs given the inputs

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmax}}\ \{\log(p(\Delta\mathbf{X}|\tilde{\mathbf{X}}, \boldsymbol{\theta}))\}. \quad (8)$$

This optimization problem is solved using off-the-shelf optimizers [27].

B. Incorporating Prior Model Knowledge

As a nonparametric approach, GP features flexible representation of the transition dynamics. However, model-based RL methods rely on training data collected from trajectory rollouts to make predictions. Therefore, the learned models do not generalize very well to unexplored regions of the state-action spaces. In this section, we consider the case when a parametric structure of the dynamics model or basis function is known, but the parameters are unknown. Incorporating explicit prior for GP regression has been introduced and discussed in [27] and [34]. Over the last decade, studies show that semiparametric models outperform purely parametric or nonparametric models for inverse and forward dynamics model identification of robotic systems [19], [35], [36]. However, these related work considered a special case when the dynamics are linear functions of the model parameters. In this paper, we consider a general case when the dynamics can be nonlinear in the parameters. First, we define the basis function $\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})$; therefore, the dynamics model can be described in a semiparametric form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \underbrace{\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})}_{\textbf{parametric}} + \underbrace{\mathbf{f}_e(\tilde{\mathbf{x}}_k)}_{\textbf{nonparametric}} \quad (9)$$

where $\mathbf{f}_e(\tilde{\mathbf{x}}_k) = \mathbf{f}(\tilde{\mathbf{x}}_k) - \boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})$ is the GP error (unmodeled) dynamics. This term represents the discrepancy between the parametric model and data sampled from the true model. We have discussed GP model learning and inference in Sections III-A1–III-A3. Given a basis function and training samples, we will learn the unknown model parameters by minimizing the loss function $l(\boldsymbol{\Theta}) = (1/2)\sum_{j=1}^{N}(\Delta\mathbf{x}_j - \boldsymbol{\phi}(\tilde{\mathbf{x}}_j, \boldsymbol{\Theta}))^2$ which is the accumulated error between the parametric model prediction and corresponding training data.

1) Parameter Estimation via Cross-Entropy Optimization: The cross-entropy (CE) approach [37] is generally used for estimating rare event probabilities via importance sampling. The key idea is to treat the optimization problem as a estimation of rare-event probabilities, which correspond to finding parameters $\boldsymbol{\Theta}$ that happens to be close to the optimal parameters $\boldsymbol{\Theta}^* = \arg\min l(\boldsymbol{\Theta})$. In contrast to gradient-based optimization methods, CE is a gradient-free method based on random sampling. The CE method performs optimization with the following basic steps: *random sampling*—draw K samples of $\boldsymbol{\Theta}$ from a Gaussian distribution. *Loss evaluation*—compute loss $l(\boldsymbol{\Theta}_i)$ for each sample i. *Sort*—sort $\boldsymbol{\Theta}_i$ by loss function values in an ascending order. *Update*—use K_l samples that correspond to low loss (in the sorted list) to update mean and variance of the distribution. Then, go back to draw samples from this updated distribution. The algorithm terminates when the variance of the distribution becomes very small. The returned solution is the mean of final sampling distribution. The CE algorithm is summarized in Algorithm 1.

Algorithm 1 CE Method for Parameter Learning

1: **Initialization:** Choose a multi-variate Gaussian distribution $\mathcal{N}(\boldsymbol{\Theta}_{\text{samp}}, \boldsymbol{\Sigma}_{\text{samp}})$
2: **Sampling:** Draw K samples of $\boldsymbol{\Theta}$ from $\mathcal{N}(\boldsymbol{\Theta}_{\text{samp}}, \boldsymbol{\Sigma}_{\text{samp}})$
3: **Loss:** Evaluate loss $l(\boldsymbol{\Theta}_i)$ for each sample i
4: **Sort:** Sort $\boldsymbol{\Theta}_{1,\ldots,K}$ w.r.t. $l(\boldsymbol{\Theta}_{1,\ldots,K})$ in ascending order
5: **Update:** Re-compute mean and variance of the sampling distribution $\boldsymbol{\Theta}_{\text{samp}} = \sum_{i=1}^{K_l}\frac{1}{K_l}\boldsymbol{\Theta}_i$ and $\boldsymbol{\Sigma}_{\text{samp}} = \sum_{i=1}^{K_l}\frac{1}{K_l}(\boldsymbol{\Theta}_i - \boldsymbol{\Theta}_{\text{samp}})(\boldsymbol{\Theta}_i - \boldsymbol{\Theta}_{\text{samp}})^{\mathsf{T}}$
6: **Iterate:** Return to step 2 until convergence
7: **Return:** Optimal estimate of parameter $\boldsymbol{\Theta}^* = \boldsymbol{\Theta}_{\text{samp}}$

2) Approximate Inference via Linearization: In order to compute the predictive distribution over $\mathbf{x}_{k+1}$ under uncertain input $p(\tilde{\mathbf{x}}_k) = (\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)$, we perform the first-order Taylor expansion of the expectation $\mathbb{E}[\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})|\tilde{\mathbf{x}}_k]$ around a reference point. Here, we consider the mean of state-action pair $\tilde{\boldsymbol{\mu}}_k$

$$\begin{aligned}\mathbb{E}_{\tilde{\mathbf{x}}_k}[\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})|\tilde{\mathbf{x}}_k] &= \mathbb{E}_{\tilde{\mathbf{x}}_k}[\boldsymbol{\phi}(\tilde{\boldsymbol{\mu}}_k, \boldsymbol{\Theta})|\tilde{\boldsymbol{\mu}}_k]\\&\quad+ \underbrace{\frac{\partial\mathbb{E}_{\tilde{\mathbf{x}}_k}[\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})|\tilde{\mathbf{x}}_k]}{\partial\tilde{\mathbf{x}}_k}\bigg|_{\tilde{\mathbf{x}}_k=\tilde{\boldsymbol{\mu}}_k}^{\mathsf{T}}}_{\boldsymbol{\Phi}_k}(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k).\end{aligned}$$

The predictive mean is obtained by evaluating the function at input mean $\tilde{\boldsymbol{\mu}}_k$. More precisely

$$\boldsymbol{\mu}_{\phi_k} \approx \boldsymbol{\phi}(\tilde{\boldsymbol{\mu}}_k, \boldsymbol{\Theta}). \tag{10}$$

Based on the linearized model, the predictive covariance is evaluated as

$$\begin{aligned}\boldsymbol{\Sigma}_{\phi_k} &= \mathbb{VAR}_{\tilde{\mathbf{x}}_k}[\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] \\ &\approx \mathbb{VAR}_{\tilde{\mathbf{x}}_k}[\boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] + \boldsymbol{\Sigma}_w \\ &= \mathbb{E}_{\tilde{\mathbf{x}}_k}[(\boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k - \boldsymbol{\Phi}_k\tilde{\boldsymbol{\mu}}_k)(\boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k - \boldsymbol{\Phi}_k\tilde{\boldsymbol{\mu}}_k)^\mathsf{T}] + \boldsymbol{\Sigma}_w \\ &= \boldsymbol{\Phi}_k\mathbb{E}_{\tilde{\mathbf{x}}_k}[(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)^\mathsf{T}]\boldsymbol{\Phi}_k^\mathsf{T} + \boldsymbol{\Sigma}_w \\ &= \boldsymbol{\Phi}_k\tilde{\boldsymbol{\Sigma}}_k\boldsymbol{\Phi}_k^\mathsf{T} + \boldsymbol{\Sigma}_w \end{aligned} \tag{11}$$

where $\boldsymbol{\Sigma}_w$ is a diagonal matrix with entries being the noise variances σ_w^2. The cross covariance between $\mathbf{x}_k$ and $\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})$ is computed as

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{\mathbf{x}}_k,\phi_k} &= \mathbb{COV}_{\tilde{\mathbf{x}}_k}[\tilde{\mathbf{x}}_k, \boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] \\ &\approx \mathbb{COV}_{\tilde{\mathbf{x}}_k}[\tilde{\mathbf{x}}_k, \boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] \\ &= \mathbb{E}_{\tilde{\mathbf{x}}_k}[(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)(\boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k - \boldsymbol{\Phi}_k\tilde{\boldsymbol{\mu}}_k)^\mathsf{T}] \\ &= \mathbb{E}_{\tilde{\mathbf{x}}_k}[(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)^\mathsf{T}]\boldsymbol{\Phi}_k^\mathsf{T} \\ &= \tilde{\boldsymbol{\Sigma}}_k\boldsymbol{\Phi}_k^\mathsf{T}. \end{aligned} \tag{12}$$

The cross covariance $\boldsymbol{\Sigma}_{\mathbf{x}_k,\phi_k} = \mathbb{COV}_{\tilde{\mathbf{x}}_k}[\mathbf{x}_k, \boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k]$ is obtained as a submatrix. The results in (10)–(12) are used to perform long-term prediction, that is

$$\begin{aligned}\boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{\mu}_{\phi_k} = \boldsymbol{\mu}_k + \boldsymbol{\phi}(\tilde{\boldsymbol{\mu}}_k, \boldsymbol{\Theta}) \\ \boldsymbol{\Sigma}_{k+1} &= \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_{\phi_k} + \boldsymbol{\Sigma}_{\mathbf{x}_k,\phi_k} + \boldsymbol{\Sigma}_{\phi_k,\mathbf{x}_k} \\ &= \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_w + \boldsymbol{\Phi}_k\tilde{\boldsymbol{\Sigma}}_k\boldsymbol{\Phi}_k^\mathsf{T} + \boldsymbol{\Phi}_k\tilde{\boldsymbol{\Sigma}}_k + \tilde{\boldsymbol{\Sigma}}_k\boldsymbol{\Phi}_k^\mathsf{T}. \end{aligned} \tag{13}$$

Different from the exact moment matching used for GP inference, the approach presented here is an approximation of the moments. In general, exact moments cannot be computed analytically because the basis function $\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})$ from a physical model is unlikely to have the same form as in GP posterior mean or variance. The partially known parametric model is very useful when predicting in the region with sparse (or no) training samples. However, the analysis above does not take into account the estimation error of the parametric model; therefore, it could still give problematic long-term predictions.

3) Semiparametric Model: Now, we combine the benefits of both parametric and nonparametric models. The semiparametric model uses the parametric part to generalize to regions in state-action space that are far from the training data, where the nonparametric part is used to model the error dynamics, i.e., difference between the true dynamics and parametric part. For instance, the posterior GP mean becomes

$$\begin{aligned}\mathbb{E}_{\mathbf{f}}[\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k] = \boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta}) + \mathbf{k}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}})(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_w\mathbf{I})^{-1} \\ \times (\Delta\mathbf{X} - \boldsymbol{\Phi}(\tilde{\mathbf{X}}, \boldsymbol{\Theta}))\end{aligned} \tag{14}$$

where $\Delta\mathbf{X} - \boldsymbol{\Phi}(\tilde{\mathbf{X}}, \boldsymbol{\Theta}) = \{\Delta\mathbf{x}_1^s - \boldsymbol{\phi}(\tilde{\mathbf{x}}_1^s, \boldsymbol{\Theta}), \ldots, \Delta\mathbf{x}_N^s - \boldsymbol{\phi}(\tilde{\mathbf{x}}_N^s, \boldsymbol{\Theta})\}$ is a collection of sampled state transition errors. It can be seen that if the query state-action pair $\tilde{\mathbf{x}}_k$ is very far away from the training samples $\tilde{\mathbf{X}}$, i.e., $\tilde{\mathbf{x}}_k - \tilde{\mathbf{x}}_i^s \to \infty$, $i = 1, \ldots, N$, the kernel $\mathbf{k}(\mathbf{x}_k, \mathbf{x}_i^s) = \sigma_f^2 \exp(-(1/2)(\mathbf{x}_k - \mathbf{x}_i^s)^T\mathbf{W}(\mathbf{x}_k - \mathbf{x}_i^s)) \to 0$. In this case, the nonparametric part almost vanishes and the parametric part dominates in making predictions. Therefore, the parametric model is essential in generalizing the model to regions in state-action space that are not covered by the training samples. This feature enables accurate prediction under insufficient exploration.

Given a Gaussian predictive distribution over the error dynamics $p(\mathbf{f}_e(\tilde{\mathbf{x}}_k)) = \mathcal{N}(\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k})$, and based on the approximate inference methods introduced in Sections III-A1, III-A2, III-B1, and III-B2, the semiparametric dynamics model can be written as

$$\begin{aligned}\boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{\mu}_{\phi_k} + \boldsymbol{\mu}_{f_k} \\ \boldsymbol{\Sigma}_{k+1} &= \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_{\phi_k} + \boldsymbol{\Sigma}_{f_k} + \boldsymbol{\Sigma}_w \\ &\quad + \boldsymbol{\Sigma}_{\mathbf{f}_k,\mathbf{x}_k} + \boldsymbol{\Sigma}_{\mathbf{x}_k,\mathbf{f}_k} + \boldsymbol{\Sigma}_{\phi_k,\mathbf{x}_k} + \boldsymbol{\Sigma}_{\mathbf{x}_k,\phi_k}\end{aligned} \tag{15}$$

where the terms $\boldsymbol{\mu}_{f_k}$ and $\boldsymbol{\Sigma}_{f_k}$ in (15) denote predictive distribution over the transition error dynamics. Note that (15) is a generalized version of (6). If $\phi(\cdot, \cdot)$ outputs zeros, (15) becomes (6). The above semiparametric formulation relies on partial knowledge of the dynamics model, which is represented by explicit basis function of the state, actions, and unknown parameters. In practice, this prior knowledge may be available for mechanical systems. Studies in the last five years have shown that semiparametric models perform substantially better than purely parametric and nonparametric models [19], [35], [36]. This advantage is more significant in real-world experiments than in simulations [19], [35]. This is because the GP error dynamics model also absorbs the modeling error of basis functions in real physical systems. In contrast, when performing experiments in simulations, the basis functions are assumed to be good representations of the dynamics and used to generate sample data. Overall, the proposed model learning and the inference scheme take into account uncertainty of unknown or partially known dynamics, which will be used for controller design in Section IV.

IV. Probabilistic Trajectory Optimization

Now, we introduce PDDP, a trajectory optimizer that employs a combination of probabilistic inference and DDP.

A. Belief Dynamics and Local Approximation

In DDP-related algorithms, a local model along a nominal trajectory $(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$ is created based on: 1) a first- or second-order approximation of the dynamics model and 2) a second-order approximation of the value function. In our proposed PDDP framework, we will create a local model along a trajectory of state distribution-control pair $(p(\bar{\mathbf{x}}_k), \bar{\mathbf{u}}_k)$. In order to incorporate uncertainty explicitly in the local model, we use the Gaussian belief $\mathbf{b}_k = [\boldsymbol{\mu}_k \ \mathrm{vec}(\boldsymbol{\Sigma}_k)]^T$ over state $\mathbf{x}_k$, where $\mathrm{vec}(\boldsymbol{\Sigma}_k)$ is the vectorization of $\boldsymbol{\Sigma}_k$. Based on (6) or (15), the belief dynamics model can be written as

$$\mathbf{b}_{k+1} = \mathcal{F}(\mathbf{b}_k, \mathbf{u}_k). \tag{16}$$

Now, we create a local model of the belief dynamics. First, we define the control and state variations $\delta\mathbf{b}_k = \mathbf{b}_k - \bar{\mathbf{b}}_k$ and $\delta\mathbf{u}_k = \mathbf{u}_k - \bar{\mathbf{u}}_k$. Next, we perform Taylor expansion of the belief dynamics around $(\mathbf{b}_k, \mathbf{u}_k)$

$$\delta\mathbf{b}_{k+1} = \mathcal{F}_k^b\delta\mathbf{b}_k + \mathcal{F}_k^u\delta\mathbf{u}_k + O(\|\delta\mathbf{b}_k\|^2 + \|\delta\mathbf{u}_k\|^2). \tag{17}$$

In this paper, we compute the first-order expansion in (17) but keep the higher order terms $O(\cdot)$ for theoretical analysis. The Jacobian matrices $\mathcal{F}_k^b$ and $\mathcal{F}_k^u$ are specified as

$$\mathcal{F}_k^b = \begin{bmatrix} \frac{\partial \boldsymbol{\mu}_{k+1}}{\partial \boldsymbol{\mu}_k} & \frac{\partial \boldsymbol{\mu}_{k+1}}{\partial \boldsymbol{\Sigma}_k} \\ \frac{\partial \boldsymbol{\Sigma}_{k+1}}{\partial \boldsymbol{\mu}_k} & \frac{\boldsymbol{\Sigma}_{k+1}}{\partial \boldsymbol{\Sigma}_k} \end{bmatrix}, \quad \mathcal{F}_k^u = \begin{bmatrix} \frac{\partial \boldsymbol{\mu}_{k+1}}{\partial \mathbf{u}_k} \\ \frac{\partial \boldsymbol{\Sigma}_{k+1}}{\partial \mathbf{u}_k} \end{bmatrix}. \quad (18)$$

The partial derivatives $(\partial \boldsymbol{\mu}_{k+1}/\partial \boldsymbol{\mu}_k)$, $(\partial \boldsymbol{\mu}_{k+1}/\partial \boldsymbol{\Sigma}_k)$, $(\partial \boldsymbol{\Sigma}_{k+1}/\partial \boldsymbol{\mu}_k)$, $(\partial \boldsymbol{\Sigma}_{k+1}/\partial \boldsymbol{\Sigma}_k)$, $(\partial \boldsymbol{\mu}_{k+1}/\partial \mathbf{u}_k)$, and $(\partial \boldsymbol{\Sigma}_{k+1}/\partial \mathbf{u}_k)$ are computed analytically. Their forms are omitted due to the space limit. For numerical implementation, the dimension of the belief can be reduced by eliminating the redundancy of $\boldsymbol{\Sigma}_k$ and the principle square root of $\boldsymbol{\Sigma}_k$ may be used for numerical robustness.

B. Optimization Criterion

Since the state is a random variable, we need to compute the expected costs and we define them as functions of the belief

$$\tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k) = \mathbb{E}_{\mathbf{x}}[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)], \quad \tilde{h}(\mathbf{b}_k) = \mathbb{E}_{\mathbf{x}}[h(\mathbf{x}_k)].$$

For instance, in many optimal control approaches, a quadratic cost function is used

$$\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) = \left(\mathbf{x}_k - \mathbf{x}_k^{\text{goal}}\right)^T \mathbf{Q}\left(\mathbf{x}_k - \mathbf{x}_k^{\text{goal}}\right) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \quad (19)$$

where $\mathbf{x}_k^{\text{goal}}$ is the target state. Given the distribution $p(\mathbf{x}_k) = \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, the expectation of original quadratic cost function is formulated as the belief-control cost

$$\tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k) = \mathbb{E}_{\mathbf{x}}[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)] = \mathcal{L}(\boldsymbol{\mu}_k, \mathbf{u}_k) + \text{tr}(\boldsymbol{\Sigma}_k \mathbf{Q}). \quad (20)$$

The cost expectation (20) scales linearly with the state covariance; therefore, it penalize both distance between the expected state and the target state, and the uncertainty of the predictive state. In this paper, we also consider a risk-sensitive cost function [38]

$$\tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k, \epsilon) = \begin{cases} \frac{1}{\epsilon} \log \mathbb{E}_{\mathbf{x}}[\exp(\epsilon \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k))], & \epsilon \neq 0 \\ \mathbb{E}_{\mathbf{x}}[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)], & \epsilon = 0 \end{cases} \quad (21)$$

where ϵ is the risk sensitivity parameter. Taking the second-order Taylor expansion of the above cost function about $\mathbb{E}[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)]$ yields [38]

$$\tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k, \epsilon) \approx \mathbb{E}_{\mathbf{x}}[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)] + \frac{\epsilon}{2} \mathbb{VAR}_{\mathbf{x}}[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)] \quad (22)$$

where the variance of cost is obtained as

$$\mathbb{VAR}_{\mathbf{x}}[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)] = \text{tr}(2\mathbf{Q}\boldsymbol{\Sigma}_k\mathbf{Q}\boldsymbol{\Sigma}_k) + 4\left(\boldsymbol{\mu}_k - \mathbf{x}_k^{\text{goal}}\right)^T \mathbf{Q}\boldsymbol{\Sigma}_k\mathbf{Q}\left(\boldsymbol{\mu}_k - \mathbf{x}_k^{\text{goal}}\right). \quad (23)$$

Partial derivatives of the cost with respect to belief and control $(\partial/\partial \mathbf{b}_k)\tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k) = \{(\partial/\partial \boldsymbol{\mu}_k)\tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k), (\partial/\partial \boldsymbol{\Sigma}_k)\tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k)\}$ and $(\partial/\partial \mathbf{u}_k)\tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k)$ can be computed analytically. Their expressions are omitted due to the space limit. The cost function (22) captures not only the expected cost, but also the predictive uncertainty of the cost. It can be viewed a generalization of the cost expectation used in RL and stochastic control. More precisely, when $\epsilon = 0$ the cost becomes *risk neutral*, which is equal to the expected cost [see (20)]. $\epsilon > 0$ corresponds to *risk averse* and $\epsilon < 0$ corresponds to *risk-seeking* criterion. In the risk-averse case, the trajectory optimizer explicitly avoids regions with high uncertainty. In the risk-seeking case, trajectories with higher variance are preferred. When no prior knowledge of the dynamics model is available, the risk-averse strategy is a good fit for the PDDP frameworks. Since it relies on local approximations of the dynamics and data sampled along one or multiple trajectories, the learned model is accurate only within the neighborhood of these trajectories. The risk-averse policies avoid wide cost distribution explicitly; therefore, extensive explorations in highly uncertainty regions are avoided, and the regions close to the sampled data are preferred. This criterion is especially beneficial for the case when very little samples are available. For the rest of this paper, the risk-sensitive learning corresponds to the risk-averse case when $\epsilon > 0$.

For a general nonquadratic cost function, we approximate it as a quadratic function along the nominal belief and control trajectory $(\bar{\mathbf{b}}, \bar{\mathbf{u}})$, that is

$$\begin{aligned} \tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k) = \tilde{\mathcal{L}}_k^0 + \left(\tilde{\mathcal{L}}_k^b\right)^{\mathsf{T}} \delta \mathbf{b}_k + \left(\tilde{\mathcal{L}}_k^u\right)^{\mathsf{T}} \delta \mathbf{u}_k \\ + \frac{1}{2} \begin{bmatrix} \delta \mathbf{b}_k \\ \delta \mathbf{u}_k \end{bmatrix}^T \begin{bmatrix} \tilde{\mathcal{L}}_k^{bb} & \tilde{\mathcal{L}}_k^{bu} \\ \tilde{\mathcal{L}}_k^{ub} & \tilde{\mathcal{L}}_k^{uu} \end{bmatrix} \begin{bmatrix} \delta \mathbf{b}_k \\ \delta \mathbf{u}_k \end{bmatrix} \\ + O(\|\delta \mathbf{b}_k\|^2, \|\delta \mathbf{u}_k\|^2) \end{aligned} \quad (24)$$

where superscripts denote partial derivatives, e.g., $\tilde{\mathcal{L}}_k^b = \nabla_b \tilde{\mathcal{L}}_k(\mathbf{b}_k, \mathbf{u}_k)$ and $\tilde{\mathcal{L}}_k^0 = \tilde{\mathcal{L}}(\bar{\mathbf{b}}_k, \bar{\mathbf{u}}_k)$. We will use this superscript rule for all cost-related terms.

C. Control Policy and Value Function Approximation

According to the dynamic programming principle [9], the Bellman equation in discrete time is specified as follows:

$$V(\mathbf{b}_k, k) = \min_{\mathbf{u}_k} \underbrace{\tilde{\mathcal{L}}(\mathbf{b}_k, \mathbf{u}_k) + V(\mathcal{F}(\mathbf{b}_k, \mathbf{u}_k), k+1)}_{Q(\mathbf{b}_k, \mathbf{u}_k)} \quad (25)$$

where $V(\mathbf{b}_k, k)$ is the value function for belief $\mathbf{b}$ at time step k. At the terminal time step, $V(\mathbf{b}_H, H) = \tilde{h}(\mathbf{b}(H))$. Given the belief dynamics (17) and cost (24), our goal is to obtain a quadratic approximation of the value function along a nominal belief trajectory $\bar{\mathbf{b}}_k$

$$V(\mathbf{b}_k, k) = V_k^0 + \left(V_k^b\right)^{\mathsf{T}} \delta \mathbf{b}_k + \frac{1}{2} \delta \mathbf{b}_k^{\mathsf{T}} V_k^{bb} \delta \mathbf{b}_k + O(\|\delta \mathbf{b}_k\|^3). \quad (26)$$

We can do so by expanding the Q-function defined in (25) along $(\bar{\mathbf{b}}_k, \bar{\mathbf{u}}_k)$

$$\begin{aligned} Q_k(\bar{\mathbf{b}}_k + \delta \mathbf{b}_k, \bar{\mathbf{u}}_k + \delta \mathbf{u}_k) \\ = Q_k^0 + \left(Q_k^b\right)^{\mathsf{T}} \delta \mathbf{b}_k + \left(Q_k^u\right)^{\mathsf{T}} \delta \mathbf{u}_k + \frac{1}{2} \begin{bmatrix} \delta \mathbf{b}_k \\ \delta \mathbf{u}_k \end{bmatrix}^T \\ \times \begin{bmatrix} Q_k^{bb} & Q_k^{bu} \\ Q_k^{ub} & Q_k^{uu} \end{bmatrix} \begin{bmatrix} \delta \mathbf{b}_k \\ \delta \mathbf{u}_k \end{bmatrix} + O\left(\left\| \begin{bmatrix} \delta \mathbf{b}_k \\ \delta \mathbf{u}_k \end{bmatrix} \right\|^2\right) \end{aligned} \quad (27)$$

where

$$\begin{aligned} Q_k^b &= \mathcal{L}_k^b + \left(\mathcal{F}_k^b\right)^{\mathsf{T}} V_{k+1}^b, \quad Q_k^u = \mathcal{L}_k^u + \left(\mathcal{F}_k^u\right)^{\mathsf{T}} V_{k+1}^b \\ Q_k^{bb} &= \mathcal{L}_k^{bb} + \left(\mathcal{F}_k^b\right)^T V_{k+1}^{bb} \mathcal{F}_k^b, \quad Q_k^{ub} = \mathcal{L}_k^{ub} + \left(\mathcal{F}_k^u\right)^T V_{k+1}^{bb} \mathcal{F}_k^b \\ Q_k^{uu} &= \mathcal{L}_k^{uu} + \left(\mathcal{F}_k^u\right)^T V_{k+1}^{bb} \mathcal{F}_k^u. \end{aligned} \tag{28}$$

In order to find the optimal control policy, we compute the local variations in control $\delta\hat{\mathbf{u}}_k$ that minimize the quadratic approximation of the Q-function in (27)

$$\begin{aligned} \delta\hat{\mathbf{u}}_k &= \arg\min_{\delta\mathbf{u}_k} [Q_k(\bar{\mathbf{b}}_k + \delta\mathbf{b}_k, \bar{\mathbf{u}}_k + \delta\mathbf{u}_k)] \\ &= \underbrace{-\left(Q_k^{uu}\right)^{-1} Q_k^u}_{\mathbf{I}_k} \underbrace{-\left(Q_k^{uu}\right)^{-1} Q_k^{ub}}_{\mathbf{L}_k} \delta\mathbf{b}_k. \end{aligned} \tag{29}$$

The optimal control can be found as $\hat{\mathbf{u}}_k = \bar{\mathbf{u}}_k + \delta\hat{\mathbf{u}}_k$. The control policy is a linear function of the belief $\mathbf{b}_k$; therefore, the controller is deterministic. To ensure convergence, we apply line search by adding a parameter $0 < \varepsilon \leq 1$ to the feedforward gain. More precisely

$$\hat{\mathbf{u}}_k = \bar{\mathbf{u}}_k + \varepsilon\mathbf{I}_k + \mathbf{L}_k\delta\mathbf{b}_k. \tag{30}$$

We start by setting $\varepsilon = 1$ and reduce it if the trajectory cost under the new control is higher than the cost of the nominal trajectory. If the new cost is lower than the nominal cost, we accept the control $\hat{\mathbf{u}}_k$ and reset $\varepsilon = 1$. We will show the global convergence of the optimal control and value function in Section IV-D. Plugging the optimal control (30) into the approximated Q-function (27) results in the following backward propagation of parameters for value function (26):

$$\begin{aligned} V_k^0 &= \mathcal{L}_k^0 + V_{k+1}^0 - \left(\frac{1}{2}\varepsilon^2 - \varepsilon\right)\left(Q_k^u\right)^{\mathsf{T}}\mathbf{I}_k \\ V_k^b &= Q_k^b + Q_k^{bu}\mathbf{I}_k, \quad V_k^{bb} = Q_k^{bb} + Q_k^{bu}\mathbf{L}_k. \end{aligned} \tag{31}$$

After the backward pass, we apply the policy (30) to generate a new control and belief trajectory by propagating the belief dynamics 16 forward in time. The belief propagation is performed using approximate inference methods, i.e., exact moment matching (see Section III-A2) and linear approximation (see Section III-B2).

D. Theoretical Analysis

In this section, we provide theoretical analysis and show that PDDP is globally convergent. The convergence property of the classical DDP has been discussed in [39] for scalar systems. However, the analysis is based on the *a priori* assumption that the algorithm would converge. Reference [40] addresses the global convergence of DDP. But it only applies when the running cost $\mathcal{L}(\cdot,\cdot) = 0$. Here, we provide a more general proof that guarantees convergence. First, we define the following vectors:

$$\begin{aligned} \Delta\mathbf{U} &= \left[\delta\mathbf{u}_1^{\mathsf{T}}, \ldots, \delta\mathbf{u}_{H-1}^{\mathsf{T}}\right]^{\mathsf{T}} \\ \mathbf{U} &= \left[\mathbf{u}_1^{\mathsf{T}}, \ldots, \mathbf{u}_{H-1}^{\mathsf{T}}\right]^{\mathsf{T}} \in \mathbb{R}^{m\times(H-1)}. \end{aligned} \tag{32}$$

In our analysis, we make the following assumptions.

Assumption 1: Q_k^{uu} is positive definite for $k = 1, \ldots, H-1$.

Assumption 2: The initial state $\mathbf{x}_1$ is known.

Remark 1: The condition in Assumption 1 can be easily ensured by applying the Levenberg–Marquardt-related trick as in [10]. The details are omitted due to page limitation. Assumption 2 is common for most episodic RL methods.

Lemma 1: For the total cost J defined in (2), belief dynamics defined in Section IV-A, and optimization criterion defined in Section IV-B, the following is true:

$$\nabla_{\mathbf{u}_k} J = \left(\mathcal{F}_k^u\right)^{\mathsf{T}}\eta_{k+1} + \mathcal{L}_k^u \tag{33}$$

for $k = 1, \ldots, H-1$, where $\eta_k = (\mathcal{F}_k^b)^{\mathsf{T}}\eta_{k+1} + \mathcal{L}_k^b$ and $\eta_H = \tilde{h}^b$.

Proof: For $k = 1, \ldots, H-1$, we have

$$\begin{aligned} &\nabla_{\mathbf{u}_k} J \\ &= \nabla_{\mathbf{u}_k}\mathbb{E}_{\mathbf{x}}\left[\sum_{\tau=k}^{H-1}\mathcal{L}(\mathbf{x}_\tau,\mathbf{u}_\tau) + h(\mathbf{x}_H)\right] \\ &= \nabla_{\mathbf{u}_k}\left[\sum_{\tau=k}^{H-1}\tilde{\mathcal{L}}(\mathbf{b}_\tau,\mathbf{u}_\tau) + \tilde{h}(\mathbf{b}_H)\right] \\ &= \frac{\partial\tilde{\mathcal{L}}(\mathbf{b}_k,\mathbf{u}_k)}{\partial\mathbf{u}_k} + \frac{\partial\tilde{\mathcal{L}}(\mathbf{b}_{k+1},\mathbf{u}_{k+1})}{\partial\mathbf{u}_k} \\ &\quad + \cdots + \frac{\partial\tilde{\mathcal{L}}(\mathbf{b}_{H-1},\mathbf{u}_{H-1})}{\partial\mathbf{u}_k} + \frac{\partial\tilde{h}(\mathbf{b}_H)}{\partial\mathbf{u}_k} \\ &= \mathcal{L}_k^u + \left(\left(\mathcal{L}_{k+1}^b\right)^{\mathsf{T}}\mathcal{F}_k^u\right)^{\mathsf{T}} + \left(\left(\mathcal{L}_{k+2}^b\right)^{\mathsf{T}}\mathcal{F}_{k+1}^b\mathcal{F}_k^u\right)^{\mathsf{T}} \\ &\quad + \cdots + \left(\left(\mathcal{L}_{H-1}^b\right)^{\mathsf{T}}\mathcal{F}_{H-2}^b\cdots\mathcal{F}_{k+1}^b\mathcal{F}_k^u\right)^{\mathsf{T}} \\ &\quad + \left((\tilde{h}^b)^{\mathsf{T}}\mathcal{F}_{H-1}^b\cdots\mathcal{F}_{k+1}^b\mathcal{F}_k^u\right)^{\mathsf{T}} \\ &= \mathcal{L}_k^u + \left(\mathcal{F}_k^u\right)^{\mathsf{T}} \\ &\quad \times \left(\underbrace{\mathcal{L}_{k+1}^b + \left(\mathcal{F}_{k+1}^b\right)^{\mathsf{T}}\left(\underbrace{\mathcal{L}_{k+2}^b + \cdots + \tilde{h}^b\mathcal{F}_{H-1}^b\cdots\mathcal{F}_{k+2}^b}_{\eta_{k+2}}\right)}_{\eta_{k+1}}\right). \end{aligned} \tag{34}$$

As a result, (33) is true. □

Next, we show that the control updates found in (30) are a descent direction.

Lemma 2: For the total cost J defined in (2) and control updates $\Delta\mathbf{U}$ defined in (30) and (32), the following is true:

$$(\nabla_{\mathbf{U}} J)^{\mathsf{T}}\Delta\mathbf{U} = -\varepsilon\sum_{k=1}^{H-1}\lambda_k + O(\varepsilon^2) \tag{35}$$

where $\lambda_k = (Q_k^u)^{\mathsf{T}}(Q_k^{uu})^{-1}Q_k^u$.

Proof: In order to show (35), it is sufficient to prove

$$\sum_{\tau=k}^{H-1}(\nabla_{\mathbf{u}_\tau} J)^{\mathsf{T}}\delta\mathbf{u}_\tau = -\varepsilon\sum_{\tau=k}^{H-1}\lambda_\tau + \left(V_k^b - \eta_k\right)^{\mathsf{T}}\delta\mathbf{b}_k + O(\varepsilon^2) \tag{36}$$

for $\tau = 1, \ldots, H-1$. It is straightforward to see that when $k = 1$, we have $\delta\mathbf{b}_1 = 0$. And the above expression is equivalent to (35). From now on, we focus on (36). First, we consider the case when $\tau = H-1$. Applying the

results from Lemma 1, the policy (30), and expressions from (28) and (31), we have

$$\begin{aligned}
&(\nabla_{\mathbf{u}_{H-1}} J)^{\mathsf{T}} \delta \mathbf{u}_{H-1} \\
&= \left(\left(\mathcal{F}^u_{H-1}\right)^{\mathsf{T}} \eta_H + \mathcal{L}^u_{H-1}\right)^{\mathsf{T}} (\varepsilon \mathbf{I}_{H-1} + \mathbf{L}_{H-1} \delta \mathbf{b}_{H-1}) \\
&= \left(Q^u_{H-1}\right)^{\mathsf{T}} (\varepsilon \mathbf{I}_{H-1} + \mathbf{L}_{H-1} \delta \mathbf{b}_{H-1}) \\
&= -\varepsilon \left(Q^u_{H-1}\right)^{\mathsf{T}} \left(Q^{uu}_{H-1}\right)^{-1} Q^u_{H-1} + \left(Q^u_{H-1}\right)^{\mathsf{T}} (\mathbf{L}_{H-1} \delta \mathbf{b}_{H-1}) \\
&= -\varepsilon \lambda_{H-1} + \left(\left(Q^u_{H-1}\right)^{\mathsf{T}} \mathbf{L}_{H-1} + \left(Q^b_{H-1}\right)^{\mathsf{T}} - \left(Q^b_{H-1}\right)^{\mathsf{T}}\right) \delta \mathbf{b}_{H-1} \\
&= -\varepsilon \lambda_{H-1} + \left(\left(V^b_{H-1}\right)^{\mathsf{T}} - \left(\left(\mathcal{F}^b_{H-1}\right)^{\mathsf{T}} \tilde{h}^b + \mathcal{L}^b_{H-1}\right)^{\mathsf{T}}\right) \delta \mathbf{b}_{H-1} \\
&= -\varepsilon \lambda_{H-1} + \left(V^b_{H-1} - \eta_{H-1}\right)^{\mathsf{T}} \delta \mathbf{b}_{H-1}. \qquad (37)
\end{aligned}$$

Therefore, (36) is true for $\tau = H-1$. Now, we assume that (36) is true for $\tau = i+1$. More precisely, we have

$$\begin{aligned}
&\sum_{\tau=i+1}^{H-1} (\nabla_{\mathbf{u}_\tau} J)^{\mathsf{T}} \delta \mathbf{u}_\tau \\
&\qquad = -\varepsilon \sum_{\tau=i+1}^{H-1} \lambda_\tau + \left(V^b_{i+1} - \eta_{i+1}\right)^{\mathsf{T}} \delta \mathbf{b}_{i+1} + O(\varepsilon^2). \qquad (38)
\end{aligned}$$

Now, we consider the case when $\tau = i$. Applying the results from Lemma 1, the policy (30), and expressions from (28) and (31), we have

$$\begin{aligned}
&\sum_{\tau=i}^{H-1} (\nabla_{\mathbf{u}_\tau} J)^{\mathsf{T}} \delta \mathbf{u}_\tau \\
&= \sum_{\tau=i+1}^{H-1} (\nabla_{\mathbf{u}_\tau} J)^{\mathsf{T}} \delta \mathbf{u}_\tau + (\nabla_{\mathbf{u}_i} J)^{\mathsf{T}} \delta \mathbf{u}_i + O(\varepsilon^2) \\
&= -\varepsilon \sum_{\tau=i+1}^{H-1} \lambda_\tau + \left(V^b_{i+1} - \eta_{i+1}\right)^{\mathsf{T}} \delta \mathbf{b}_{i+1} + \left(\eta^{\mathsf{T}}_{i+1} \mathcal{F}^u_i + \left(\mathcal{L}^u_i\right)^{\mathsf{T}}\right) \\
&\quad \times \delta \mathbf{u}_i + O(\varepsilon^2) \\
&= -\varepsilon \sum_{\tau=i+1}^{H-1} \lambda_\tau + \left(V^b_{i+1} - \eta_{i+1}\right)^{\mathsf{T}} \\
&\quad \times \left(\mathcal{F}^b_i \delta \mathbf{b}_i + \mathcal{F}^u_i \delta \mathbf{u}_i + O(\|\delta \mathbf{u}_i\|^2 + \|\delta \mathbf{b}_i\|^2)\right) \\
&\quad + \left(\eta^{\mathsf{T}}_{i+1} \mathcal{F}^u_i + \left(\mathcal{L}^u_i\right)^{\mathsf{T}}\right) \delta \mathbf{u}_i + O(\varepsilon^2) \\
&= -\varepsilon \sum_{\tau=i+1}^{H-1} \lambda_\tau + \left(V^b_{i+1} - \eta_{i+1}\right)^{\mathsf{T}} \mathcal{F}^b_i \delta \mathbf{b}_i \\
&\quad + \left(\left(\mathcal{F}^u_i\right)^{\mathsf{T}} V^b_{i+1} + \mathcal{L}^u_i\right)^{\mathsf{T}} \delta \mathbf{u}_i + O(\varepsilon^2) \\
&= -\varepsilon \sum_{\tau=i+1}^{H-1} \lambda_\tau + \left(V^b_{i+1} - \eta_{i+1}\right)^{\mathsf{T}} \mathcal{F}^b_i \delta \mathbf{b}_i + \left(Q^u_i\right)^{\mathsf{T}} \\
&\quad \times \left(-\varepsilon \left(Q^{uu}_i\right)^{-1} Q^u_i - \left(Q^{uu}_i\right)^{-1} Q^{ub}_i \delta \mathbf{b}_i\right) + O(\varepsilon^2) \\
&= -\varepsilon \sum_{\tau=i+1}^{H-1} \lambda_\tau - \varepsilon \left(Q^u_i\right)^{\mathsf{T}} \left(Q^{uu}_i\right)^{-1} Q^u_i + \left(Q^b_i - \mathcal{L}^b_i - \eta_i + \mathcal{L}^b_i\right)^{\mathsf{T}} \\
&\quad \times \delta \mathbf{b}_i - \left(Q^u_i\right)^{\mathsf{T}} \left(Q^{uu}_i\right)^{-1} Q^{ub}_i \delta \mathbf{b}_i + O(\varepsilon^2) \\
&= -\varepsilon \sum_{\tau=i+1}^{H-1} \lambda_\tau - \varepsilon \lambda_i + \left(Q^b_i - \left(Q^u_i\right)^{\mathsf{T}} \left(Q^{uu}_i\right)^{-1} Q^{ub}_i\right) - \eta_i\Big)^{\mathsf{T}} \delta \mathbf{b}_i \\
&\quad + O(\varepsilon^2) \\
&= -\varepsilon \sum_{\tau=i}^{H-1} \lambda_\tau + \left(V^b_i - \eta_i\right)^{\mathsf{T}} \delta \mathbf{b}_i + O(\varepsilon^2) \qquad (39)
\end{aligned}$$

where we have used the fact that $\delta \mathbf{b}_k = O(\varepsilon)$ and $\delta \mathbf{u}_k = O(\varepsilon)$ [40]. Now, we complete the proof of (36) for $k = i$. By induction, (36) is true for $k = 1, \ldots, H-1$, and hence (35) is true. □

Remark 2: Lemma 2 implies for sufficiently small ε and nonzero λ_k, and the control update $\delta \mathbf{u}_k$ is a descent direction along the nominal trajectory.

Next, we provide an expression for the variation of the total cost after using the optimal control policy (30).

Lemma 3: The total cost variation at each iteration is

$$\Delta J = \left(\frac{1}{2}\varepsilon^2 - \varepsilon\right) \sum_{k=1}^{H-1} \lambda_k + O(\varepsilon^3) \qquad (40)$$

where $\lambda_k = (Q^u_k)^{\mathsf{T}} (Q^{uu}_k)^{-1} Q^u_k$.

Proof: First, let $(\bar{\mathbf{b}}, \bar{\mathbf{u}})$ be the nominal state-control trajectory at iteration i. We denote the total cost at iteration i (along the nominal trajectory) by

$$J^{(i)} = \sum_{k=1}^{H-1} \tilde{\mathcal{L}}(\bar{\mathbf{b}}_k, \bar{\mathbf{u}}_k) + \tilde{h}(\bar{\mathbf{b}}_H) \qquad (41)$$

and denote the cost at iteration $i+1$ (after applying the policy) [see (30)] by $J^{(i+1)}$. In particular, $J^{(i+1)}$ will be equal to V^0_k in (31), plus some extra terms which appear when higher order expansions of Q are considered in (27). One can obtain

$$\begin{aligned}
J^{(i+1)} &= V(\mathbf{b}_1, 1) = V^0_1 + O(\varepsilon^3) \\
&= \tilde{\mathcal{L}}(\bar{\mathbf{b}}_1, \bar{\mathbf{u}}_1) + \left(\frac{1}{2}\varepsilon^2 - \varepsilon\right) \lambda_1 + V(\bar{\mathbf{b}}_2) + O(\varepsilon^3) \\
&= \tilde{\mathcal{L}}(\bar{\mathbf{b}}_1, \bar{\mathbf{u}}_1) + \left(\frac{1}{2}\varepsilon^2 - \varepsilon\right) \lambda_1 + \tilde{\mathcal{L}}(\bar{\mathbf{b}}_2, \bar{\mathbf{u}}_2) \\
&\quad + \left(\frac{1}{2}\varepsilon^2 - \varepsilon\right) \lambda_2 + V(\bar{\mathbf{b}}_3) + O(\varepsilon^3) \\
&= \sum_{k=1}^{H-1} \tilde{\mathcal{L}}(\bar{\mathbf{b}}_k, \bar{\mathbf{u}}_k) + \tilde{h}(\bar{\mathbf{b}}_H) + \left(\frac{1}{2}\varepsilon^2 - \varepsilon\right) \\
&\quad \times (\lambda_1 + \cdots + \lambda_{H-1}) + O(\varepsilon^3) \\
&= J^{(i)} + \left(\frac{1}{2}\varepsilon^2 - \varepsilon\right) \sum_{k=1}^{H-1} \lambda_k + O(\varepsilon^3). \qquad (42)
\end{aligned}$$

Therefore, $\Delta J = J^{(i+1)} - J^{(i)} = ((1/2)\varepsilon^2 - \varepsilon) \sum_{k=1}^{H-1} \lambda_k + O(\varepsilon^3)$ which concludes the proof. □

Remark 3: Based on (40), a valid candidate for ε in (30) should satisfy at each iteration the following line search condition:

$$J^{(i+1)} - J^{(i)} \leq -\kappa \varepsilon \sum_{k=1}^{H-1} \lambda_k \qquad (43)$$

where κ is a small positive number (e.g., $\kappa \leq 0.01$).

Now, we show the convergence of the proposed PDDP algorithm.

Theorem 1: As the iteration number approaches infinity, the total cost J and control sequence $\mathbf{U}$ converge to a stationary point for arbitrary initialization.

Proof: From the fact $0 < \varepsilon \leq 1$, it is straightforward to see

$$\varepsilon \leq 1 \Longrightarrow \left(\frac{1}{2}\varepsilon - 1\right) \leq -\frac{1}{2} \Longrightarrow \left(\frac{1}{2}\varepsilon^2 - \varepsilon\right) \leq -\frac{1}{2}\varepsilon \quad (44)$$

therefore, from Lemma 3, we have the cost reduction at the ith iteration

$$\Delta J^{(i)} \leq -\frac{1}{2}\varepsilon \sum_{k=1}^{H-1} \left(Q_k^u\right)^{\mathsf{T}} \left(Q_k^{uu}\right)^{-1} Q_k^u + O(\varepsilon^3). \quad (45)$$

From Lemma 2, there exists an $\varepsilon_1 \leq 1$ such that

$$(\nabla_{\mathbf{U}} J)^{\mathsf{T}} \Delta \mathbf{U} = -\varepsilon \sum_{k=1}^{H-1} \left(Q_k^u\right)^{\mathsf{T}} \left(Q_k^{uu}\right)^{-1} Q_k^u \quad \forall \varepsilon \in (0, \varepsilon_1]. \quad (46)$$

Therefore, the control update (29) is a descent direction. In addition, from (45), there exists an $\varepsilon_2 \in (0, \varepsilon_1]$ such that

$$\Delta J^{(i)} \leq -\frac{1}{2}\varepsilon \sum_{k=1}^{H-1} \left(Q_k^u\right)^{\mathsf{T}} \left(Q_k^{uu}\right)^{-1} Q_k^u \quad \forall \varepsilon \in (0, \varepsilon_2] \quad (47)$$

which indicates that $J^{(i)}$ is monotonic decreasing for arbitrary initialization. To proceed, we assume that the search space is compact. Hence, since J is a continuous function of $\mathbf{U}$, there exists a control sequence $\mathbf{U}^*$ such that

$$\lim_{i \to \infty} J^{(i)} = J(\mathbf{U}^*). \quad (48)$$

Therefore, the total cost J is convergent. In addition, (48) implies that as $i \to \infty$, for arbitrary initialization and $k = 1, \ldots, H-1$

$$\begin{aligned} &\Delta J^{(i)} \to 0 \Longrightarrow \left(Q_k^u\right)^{\mathsf{T}} \left(Q_k^{uu}\right)^{-1} Q_k^u \to 0 \\ &Q_k^{uu} \text{ is p.d.} \Longrightarrow \left(Q_k^{uu}\right)^{-1} Q_k^u \to \mathbf{0} \end{aligned} \quad (49)$$

where $\mathbf{0} = [0, \ldots, 0]^{\mathsf{T}} \in \mathbb{R}^{m \times 1}$. The above condition indicates that the feedforward policy $\mathbf{I}_k$ defined in (29) vanishes as $i \to \infty$. Hence, $\delta \mathbf{u}_k \to \mathbf{L}_k \delta \mathbf{b}_k$. The initial state is given so

$$\begin{aligned} \delta \mathbf{b}_1 = \mathbf{0} &\Longrightarrow \delta \mathbf{u}_1 \to \mathbf{L}_1 \delta \mathbf{b}_1 \to \mathbf{0} \\ &\Longrightarrow \delta \mathbf{b}_2 = \mathcal{F}_1^b \delta \mathbf{b}_1 + \mathcal{F}_1^u \delta \mathbf{u}_1 + O(\|\delta \mathbf{b}_1\|^2 + \|\delta \mathbf{u}_1\|^2) \to \mathbf{0} \\ &\Longrightarrow \delta \mathbf{u}_2 \to \mathbf{L}_2 \delta \mathbf{b}_2 \to \mathbf{0}. \end{aligned} \quad (50)$$

It is straightforward to extend this analysis and we have $\delta \mathbf{u}_k \to \mathbf{0}$ for $k = 1, \ldots, H-1$. Therefore, the controls will converge to $\mathbf{U}^*$.

Finally, we show that $\mathbf{U}^*$ is a stationary point. First note that (49) implies that $Q_k^u \to \mathbf{0}$ for all time instants. As a consequence, $Q_k^b = V_k^b$. By using the expression in Lemma 1, we have

$$\begin{aligned} \nabla_{u_{H-1}^*} J &= \left(\mathcal{F}_{H-1}^u\right)^{\mathsf{T}} \tilde{h}^b + \mathcal{L}_{H-1}^u = Q_{H-1}^u \to \mathbf{0} \\ \nabla_{u_{H-2}^*} J &= \left(\mathcal{F}_{H-2}^u\right)^{\mathsf{T}} \eta_{H-1}^b + \mathcal{L}_{H-2}^u \\ &= \left(\mathcal{F}_{H-2}^u\right)^{\mathsf{T}} \underbrace{\left(\left(\mathcal{F}_{H-1}^b\right)^{\mathsf{T}} \tilde{h}^b + \mathcal{L}_{H-1}^b\right)}_{Q_{H-1}^b = V_{H-1}^b} + \mathcal{L}_{H-2}^u \\ &= Q_{H-2}^u \to \mathbf{0}. \end{aligned} \quad (51)$$

We continue the above analysis backward in time for $\nabla_{u_{H-3}^*} J \to 0, \ldots, \nabla_{u_1^*} J \to 0$ hence $\nabla_{\mathbf{U}^*} J \to 0$. It is deduced that the controls converge to a stationary point $\mathbf{U}^*$ for arbitrary initialization. □

Remark 4: The above analysis shows that our method converges to a stationary solution under some mild assumptions. Moreover, an explicit expression for the decrease in the cost after each iteration is provided [see (40)]. Note that global optimality of the solution to the original problem is not guaranteed here for two reasons: 1) the nonlinear belief dynamics lead to a nonconvex optimization problem and 2) the belief is a Gaussian approximation and not necessarily the true posterior distribution, and we have discussed this problem in Section III-A2.

Remark 5: Our method can be classified as a second-order optimal control method and, therefore, is expected to outperform standard first-order gradient-based methods [22]. Moreover, under the aforementioned assumptions, PDDP is also capable of achieving locally quadratic convergence rates, i.e., as we get closer to the stationary solution, there exists a constant $c > 0$ such that

$$\|\mathbf{U}^{(i+1)} - \mathbf{U}^*\| \leq c \|\mathbf{U}^{(i)} - \mathbf{U}^*\|^2. \quad (52)$$

This has been proven for scalar systems with no terminal cost term in [39] and generic problems in [41]. We refer the reader to these papers for more details, since the same proof can be used here. Note that this extension requires second-order expansions of the belief dynamics (16), which would add an extra term in Q^{bb}, Q^{bu}, and Q^{uu} 28. Nevertheless, in this paper, we choose to use only the first-order approximation of the belief dynamics (16). Empirically, neglecting the Hessians of the dynamics results in a more computationally efficient algorithm without sacrificing the quality of solution (see [42] for discussion).

E. Control Constraint

Control constrains can be taken into account in different fashions, and it has been shown that using naive clamping and squashing functions performs unfavorably compared to directly incorporating the constraints while minimizing the Q-function [43]. In this paper, we take into account the control constraints by solving a quadratic programming (QP) problem subject to a box constraint

$$\begin{aligned} &\min_{\delta \mathbf{u}_k} \; Q_k(\mathbf{b}_k + \delta \mathbf{b}_k, \mathbf{u}_k + \delta \mathbf{u}_k) \\ &\text{s.t. } \mathbf{u}_{\min} \leq \mathbf{u}_k + \delta \mathbf{u}_k \leq \mathbf{u}_{\max} \end{aligned} \quad (53)$$

where $\mathbf{u}_{\min}$ and $\mathbf{u}_{\max}$ correspond to the lower and upper bounds of the controller. The QP problem (53) can be solved efficiently due to the fact that at each time step, the scale of the QP problem is relatively small. In addition, warm-start can be used to further speed up the optimization in the backward pass. Solving (53) directly is not feasible, since $\delta \mathbf{b}$ is not known in the backward sweep. The optimum consists of feedforward and feedback parts, i.e., $\mathbf{I}_k + \mathbf{L}_k \delta \mathbf{b}_k = \arg\min_{\delta \mathbf{u}_k} [Q_k(\mathbf{b}_k + \delta \mathbf{b}_k, \mathbf{u}_k + \delta \mathbf{u}_k)]$, here we adopt the strategy in [43] using the projected-Newton algorithm [44]. The feedforward gain is computed by solving the QP problem

$$\begin{aligned} \mathbf{I}_k = \arg\min_{\delta \mathbf{u}_k} &\left[\delta \mathbf{u}_k^{\mathsf{T}} Q_k^{uu} \delta \mathbf{u}_k + Q_k^b \delta \mathbf{u}_k\right] \\ &\text{s.t. } \mathbf{u}_{\min} \leq \mathbf{u}_k + \delta \mathbf{u}_k \leq \mathbf{u}_{\max}. \end{aligned} \quad (54)$$

The algorithm gives the decomposition of $Q_k^{uu} = \begin{bmatrix} Q_{k,ff}^{uu} & Q_{k,fc}^{uu} \\ Q_{k,cf}^{uu} & Q_{k,cc}^{uu} \end{bmatrix}$, where the indices f, c correspond to clamped (when $\mathbf{u}_k = \mathbf{u}_{\min}$ or $\mathbf{u}_{\max}$) or free ($\mathbf{u}_{\min} < \mathbf{u}_k < \mathbf{u}_{\max}$) parts, respectively. The feedback gain associated with the free part is obtained by $\mathbf{L}_k = -Q_{k,ff}^{uu} Q_k^{ub}$. The rows of $\mathbf{L}_k$ corresponding to clamped controls are set to be zero. An example of the constrained and unconstrained controllers for the quadrotor tasks is shown in Fig. 3.

F. Summary of Algorithm

The PDDP framework for RL can be summarized in Algorithm 2. From a high-level view, the main loop of this algorithm consists of three basic components.

Algorithm 2 PDDP for RL

1: **Initialization:** Apply random or pre-specified control inputs to the physical system (1). Collect data.
2: **repeat** ▷ *Main reinforcement learning loop*
3: **Model learning:** Learn GP hyper-parameters $\boldsymbol{\theta}$ by evidence maximization (8) given sample data. Optimize dynamics model parameters if necessary (see section III-B).
4: **repeat** ▷ *Probabilistic trajectory optimization loop*
5: **Local approximation:** Obtain linear approximation of the belief dynamics (17) and quadratic approximation of the cost (24) along a nominal trajectory $(\bar{\mathbf{b}}_k, \bar{\mathbf{u}}_k)$. See section IV-A.
6: **Backward sweep:** Compute the approximation of the value function (31) and obtain optimal policy for control correction $\delta\hat{\mathbf{u}}_k$ (29). See section IV-C.
7: **Forward sweep:** Update control $\bar{\mathbf{u}}_k$ and perform approximate inference to obtain a new nominal trajectory $(\bar{\mathbf{b}}_k, \bar{\mathbf{u}}_k)$. See sections III-A2, III-B2.
8: **until** Termination condition is satisfied
9: **return** Optimal trajectory $(\bar{\mathbf{b}}_k, \bar{\mathbf{u}}_k)$ and control policy
10: **Interaction:** Apply the optimized control policy $\mathbf{L}_k$ to the system (1) along the optimized trajectory $\bar{\mathbf{u}}_k$ and record data. Additional rollouts can be generated using variations of the learned controller.
11: **until** Task learned
12: **Return:** Optimal trajectory and control policy

1) *Model Learning (Step 3):* Learning a probabilistic GP dynamics model using data sampled from interactions with physical systems. If prior knowledge (basis function) of the system dynamics is given, model parameters may be learned as discussed in Section III-B1.
2) *Probabilistic Trajectory Optimization (Steps 5–7):* This iterative scheme has three steps at each iteration. First (step 5), we compute the linear approximation (17) of the belief dynamics (16) and quadratic approximation of the cost (24) along a nominal trajectory. Second (step 6), we compute a local optimal control policy (30) by backward propagation of the value function (31). Control constraints can be incorporated in the backward propagation (see Section IV-E). In addition we employ a line search strategy to ensure convergence (see Section IV-C for details). Third (step 7), we update and apply the control to obtain a new belief trajectory using approximate inference methods introduced in Sections III-A2 and III-B2. This trajectory is used as the nominal for the next iteration. Regarding the termination condition (step 8), we consider three stopping criteria.
   a) Maximum iteration number is reached.
   b) Policy improvement is small enough.
   c) Predictive variance exceeds a threshold.

   The optimization techniques stop the iteration process when at least one of the corresponding termination criteria is satisfied.
3) Interaction (Step 10): In order to collect new state and control samples, we apply the optimized trajectory and control policy to the physical systems and generate a trajectory rollout. Additional trajectories can be generated using the variations of the optimized trajectory.

G. Computational Complexity

1) Approximate Inference: The major computational effort is devoted to GP inferences. In particular, the complexity of one-step moment matching (see Section III-A) is $\mathcal{O}((N)^2 n^2 (n+m))$ [22], which is fixed during the iterative process of PDDP. We found that a small number of sampled trajectories ($N \leq 5$) are able to provide good performances for a system of moderate size (6–12 state dimensions). However, for higher dimensional problems, sparse or local approximation of GP [18], [45], [46] may be used to reduce the computational cost of GP dynamics propagation.

2) Controller Learning: According to (29), learning policy parameters $\mathbf{l}_k$ and $\mathbf{L}_k$ require computing the inverse of $\mathbf{Q}_k^{uu}$, which has the computational complexity of $\mathcal{O}(m^3)$, where m is the dimension of control input. For the case of control constrained, the QP also leads to the complexity of $\mathcal{O}(m^3)$. It comes from the Cholesky factorization of the projected Newton solver. As a local trajectory optimization method, PDDP has a comparable scalability to the classical DDP.

H. Relation to Existing Works

Here, we summarize the novel features of PDDP in comparison with some notable DDP-related frameworks for stochastic systems (see also Table I). First, PDDP is inherently different from iLQG [10] and sDDP [11], in which the dynamics model is known, and model uncertainty is ignored. Second, PDDP shares some similarities with the belief space iLQG [42] framework, which performs the belief space trajectory optimization based on an extended Kalman filter. Belief space iLQG assumes that a dynamics model is given and the stochasticity comes from the process noises. PDDP, however, is a data-driven approach that learns the dynamics models and controls from data, and it takes into account model uncertainties using GPs. Third, PDDP is also comparable with iLQG-LD [16], which is based on learned dynamics using locally weighted projection regression. The minimax DDP [13] uses a similar model learning approach (RFWR)

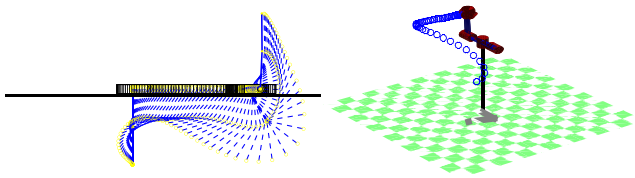
TABLE I
COMPARISON WITH DDP-RELATED FRAMEWORKS

	PDDP (this paper)	**Belief space iLQG**[42]/**sDDP**[50]	**iLQG-LD**[16]	**iLQG**[10], **sDDP**[11]	**AGP-iLQR**[26]	**Minimax DDP**[13, 51]
Optimization variables	belief $\boldsymbol{\mu}, \boldsymbol{\Sigma}$, control $\mathbf{u}$	belief $\boldsymbol{\mu}, \boldsymbol{\Sigma}$, control $\mathbf{u}$	state $\mathbf{x}$, control $\mathbf{u}$	state $\mathbf{x}$, control $\mathbf{u}$	mean of state $\boldsymbol{\mu}$, control $\mathbf{u}$	state $\mathbf{x}$, control $\mathbf{u}$
Optimization criterion	Cost distribution	Expected cost	Expected cost	Expected cost	Expected cost and predictive state variance	Cost incorporates a disturbance term
Dynamics model	Unknown (GP) or partially known (GP+parametric model)	Known	Unknown (LWPR)	Known	Unknown (Approximate GP)	Partially known (RFWR)
Linearization	Analytic Jacobian	Finite differences	Analytic Jacobian	Finite differences	Analytic Jacobian	Analytic Jacobian

with partially known dynamics. These methods do not incorporate model uncertainty and, therefore, require a relatively large amount of data to learn an accurate model. Furthermore, PDDP does not suffer from the high computational cost of finite differences used to numerically compute the first-order expansions [10], [42] and the second-order expansions [11] of the underlying dynamics. PDDP computes Jacobian matrices analytically [see (18)]. Furthermore, a recent method AGP-iLQR [26] shows impressive performance and also uses GPs as a dynamics model. However, it drops the uncertainty when performing multistep predictions. In contrast, PDDP takes into account explicit uncertainty and optimizes with respect to the belief over state. In terms of optimization criterion, AGP-iLQR incorporates predictive variance of the state transition, and this term appears linearly in the cost function. PDDP utilizes the distribution over cost (expectation and variance) as the performance criterion, which is more general. In addition, the proposed approach is a significant extension of the preliminary work on PDDP [47]. For instance, in this paper, we develop a semiparametric learning scheme by incorporating prior model knowledge; we explore risk-sensitive learning using predicted cost variance; and we take into account control constraints by solving a QP problem (see Section IV-E). All of these extensions substantially improve the applicability of the PDDP framework.

PDDP is also related to other RL frameworks that are not based on local trajectory optimization. The most notable algorithm is PILCO [22], a model-based policy search approach using GPs. Their main differences can be summarized as follows: first, PDDP is a self-contained trajectory optimization method that features fast convergence, in contrast, PILCO requires an extra optimizer for policy improvement (e.g., BFGS). Second, PILCO requires designs of policy parameterization and solves relatively higher dimensional optimization problems (depending on the number of parameters), while PDDP does not require any policy parameterization. Third, PDDP uses a forward–backward sweep scheme, and the policy improvement at time step k takes into account improvement at future steps. In contrast, PILCO as well as most policy gradient methods find time-independent policies, which is less efficient.

Many other GP-based planning and Bayesian RL algorithms focus on discrete domains or finite state/action spaces.



(a) (b)

Fig. 1. CDIP and Puma-560 tasks.

These works usually provide an error bound for their approximations [48], [49]. Our problem of interest is inherently different, since we focus on continuous domains.

V. EXPERIMENTAL EVALUATION

In this section, we evaluate the PDDP framework using three nontrivial simulated examples. We demonstrate the performance of PDDP by comparative analyses. All experiments were performed in MATLAB on a 3.7-GHz Intel i7 PC.

A. Tasks

In this paper, we consider three simulated tasks: cart-double inverted pendulum (CDIP) swing-up, PUMA-560 robotic arm reaching, and quadrotor flight.

1) Cart-Double Inverted Pendulum Swing-Up: CDIP swing-up is a challenging control problem, because the system is highly underactuated with three degrees of freedom and only one control input. The system has six state-dimensions (cart position/velocity, link 1, two angles, and angular velocities). The physical model parameters are: masses of the cart and two links, lengths of two links, and the coefficient of friction. The goal of the swing-up problem is to find a sequence of control input to force both pendulums from initial position (π, π) to the inverted position $(2\pi, 2\pi)$. The balancing task requires the velocity of the cart and the angular velocities of both pendulums to be zero. The time horizon for the task is 1.2 s and $dt = 0.02$. We sample five trajectories for each optimization stage. For optimization criterion, we use both the expected (risk-neutral) cost and the risk-sensitive cost with $\epsilon = 0.2$. The CDIP task posture is shown in Fig. 1(a).

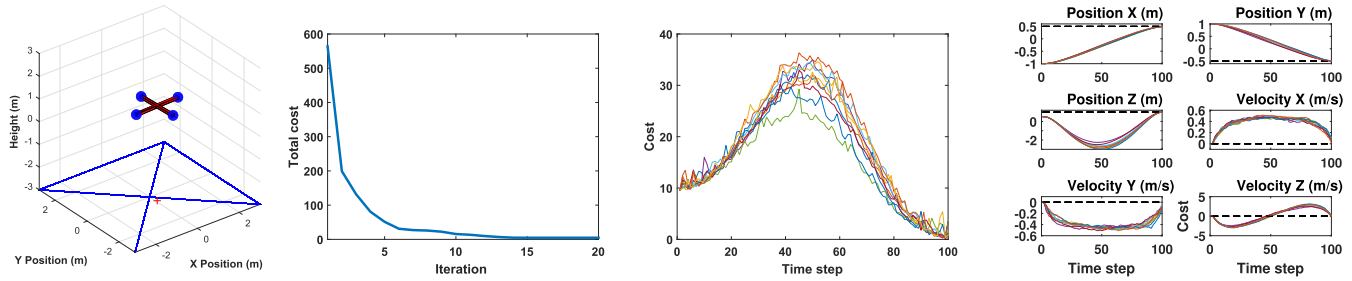


Fig. 2. Quadrotor flight task. (a) Quadrotor simulation environment. (b) Expected trajectory cost reduction during PDDP optimization. (c) Trajectory costs over ten independent trials using the optimized control policy. (d) State trajectories of the quadrotor task collected over ten independent trials using the optimized controller. Dashed lines: desired states.

2) PUMA-560 Robotic Arm Reaching: PUMA-560 is a 3-D robotic arm that has 12 state dimensions and six degrees of freedom with six actuators on the joints. The physical model parameters are the moments of inertia, masses, centers of gravity, lengths, and offsets for six links. The task is to steer the end-effector to the desired position and orientation. See an illustration in Fig. 1(b). The time horizon for the task is 2 s and $dt = 0.02$. We sample three trajectories for each optimization stage. For optimization criterion, we use the expected cost.

3) Quadrotor Flight: Quadrotors are underactuated rotorcraft, which rely on symmetry in order to fly in a conventional, stable flight regime. With six degrees of freedom and four rotors to control them, the systems attitude is highly coupled with its 3-D movement. See Fig. 2(a) for an example. The objective is to start at (1, 1, 0.5) and finish at position (0.5, 1, 1.5) after 4 s. All velocities and angles begin at zero and should end at zero—thus the quadrotor begins at rest at the start position and should reach the goal position and stop there. The controls are thrust forces of the four rotors and we consider the control constraint $\mathbf{u}_{\min} = 0.5$ and $\mathbf{u}_{\max} = 3$. A comparison between the constrained and unconstrained controllers is shown in Fig. 3. For optimization criterion, we use the risk-sensitive cost with $\epsilon = 0.5$. We assume that the partial knowledge of the dynamics model is known, i.e., the structure of the transition dynamics with the following unknown parameters: moments of inertia about X-, Y-, Z-axes, the distance of rotor to the center of mass, and the mass of the quadrotor. Physical model parameters were learned using five sample rollouts and another five rollouts were used for GP model learning. See Section III-B for details regarding semiparametric model learning and inference.

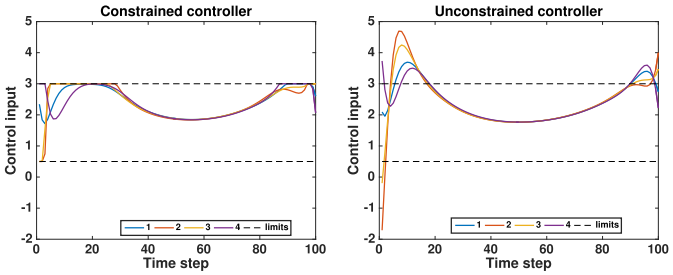


Fig. 3. (a) Constrained and (b) unconstarined control inputs (four motor torques) for the quadrotor task. The constraint is incorporated by solving a QP (see Section IV-E).

B. Data Efficiency

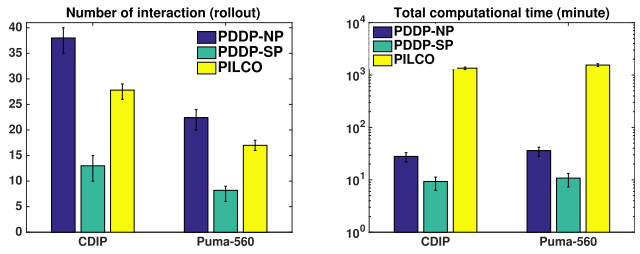
For the case of unknown dynamics, both PDDP and PILCO are based on nonparametric GP dynamics models. As shown in Fig. 4(a), PDDP performs slightly worse than PILCO in terms of data efficiency based on the number of interactions required to learn the same tasks. The number of interactions indicates the total amount of sample rollouts required from the physical systems. Possible reasons for the slightly worse performances are: 1) PDDP's policy is linear which might be restrictive, while PILCO yields nonlinear policy parameterizations and 2) as a global method, PILCO encourages more exploration especially in the early stages of learning. In contrast, PDDP is a local method, and it searches for solution in the neighborhoods of trajectories. Despite these differences, PDDP offers close performances compared to PILCO in terms of data efficiency with less computational cost. According to the analysis in [22], PDDP would outperform most RL algorithms by at least an order of magnitude in terms of data efficiency.

C. Computational Efficiency

In terms of total computational time required to obtain the final controller, PDDP outperforms PILCO significantly, as shown in Fig. 4(b). PILCO requires an iterative method (e.g., CG or BFGS) to solve global and high-dimensional optimization problems (depending on the policy parameterization). In contrast, PDDP successively computes local optimal controls (29) without an extra optimizer and features fast convergence [39]. The major computational demand of PDDP comes from the approximate inference in GPs (moment matching). We did not use any approximate GP methods such as local GP [18], [52] or sparse GP [45], [46], [53], [54] approximations. These methods can be applied in PDDP to speed up computations when the training data size is large [55].

D. Nonparametric Versus Semiparametric Learning

As shown in Fig. 4, PDDP based on semiparametric model learning and inference (with known basis function)



(a) (b)

Fig. 4. Comparison between PDDP and PILCO in terms of (a) number of interactions with the physical system and (b) total computational time required for learning the CDIP and Puma-560 tasks. PDDP-NP and PDDP-SP correspond to the nonparametric and semiparametric cases, respectively. The results were averaged over five independent trials.

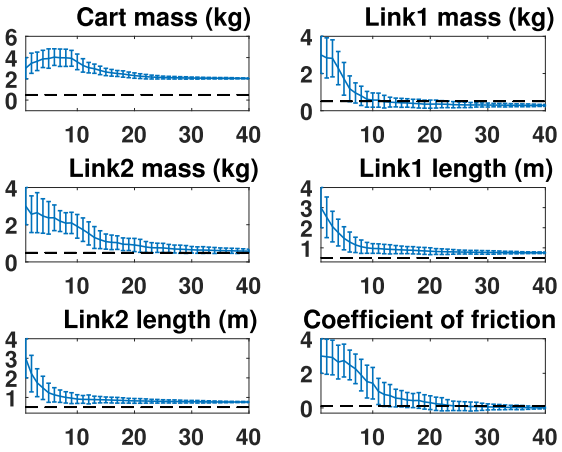
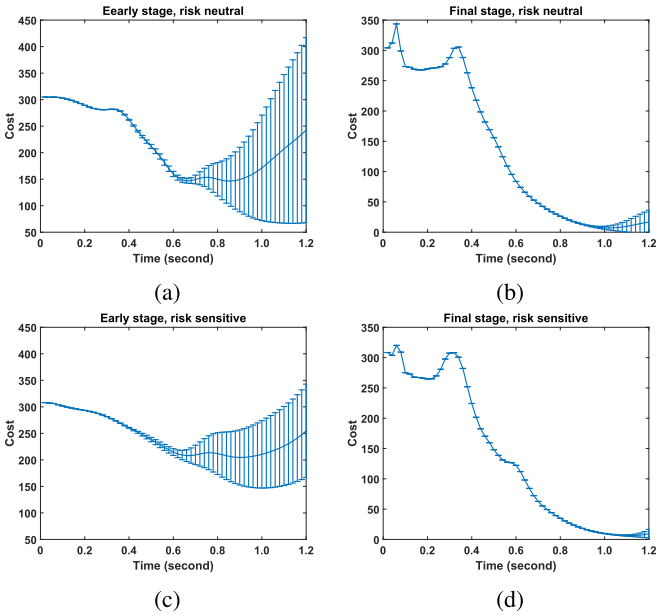


Fig. 5. CE parameter optimization for a CDIP dynamics model with bad initial guesses. Horizontal axis: iteration number. Vertical axis: parameter value. Dashed lines: true parameter values. Error bars: mean and variance of sampling distributions at each iteration. Note that some parameters converge to the true values while some others converge to local minima.

outperforms both PDDP and PILCO under unknown dynamics in terms of data and computational efficiency. To distinguish the effect of parametric and nonparametric models, at each learning stage, we use the old samples from previous stages for parameter estimation. The purpose of the parametric model is to generalize to the state-action space regions that are far away from the training data used for nonparametric model learning. This result is not surprising, but it shows the potential of applying PDDP in a more practical way. In real-world applications, these basis functions are usually available for mechanical/robotics systems from physics-based models. Learning the model from scratch is conceptually appealing but practically challenging when the available data are not sufficiently informative. In addition, the semiparametric approach takes into account the uncertainty of parametric modeling error. For instance, Fig. 5 shows the physical model parameter learning performance for the CDIP task using Algorithm 1. Due to data insufficiency, some parameter estimations deviate from the true values. This parametric error is learned as the GP error dynamics model (see Section III-B). The semiparametric PDDP could be applied to more challenging tasks compared with purely nonparametric methods. For example, the quadrotor system has highly coupled outputs. In contrast,



(a) (b) (c) (d)

Fig. 6. Comparison of the predicted trajectory cost distributions between (a) and (b) risk-neutral and (c) and (d) risk-sensitive learning via PDDP. Early stage and final stage correspond to the predictions after one and eight optimization stages.

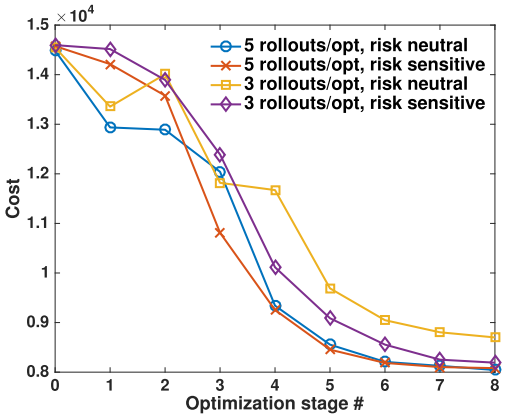


Fig. 7. Total trajectory costs by applying PDDP policy after each optimization stage for risk-neutral ($\epsilon = 0$) and risk-sensitive learning ($\epsilon = 0.2$). Three and five rollouts were used for learning in both cases.

the nonparametric GP assumed independent output for each dimension and, therefore, has limited modeling power in this case. We use the semiparametric PDDP to successfully learn a policy in a single optimization stage. Results for policy learning and testing are shown in Fig. 2. The optimization takes less than 20 iterations [see Fig. 2(b)].

E. Risk-Sensitive Versus Risk-Neutral Learning

Another distinct feature for PDDP is that it incorporates the probability distribution over the cost as the performance criterion (see Section IV-B), more precisely the mean and variance of the predicted cost. We evaluate the effect of the cost function by comparing the risk-sensitive and risk-neutral cases for the CDIP task. Fig. 6 shows the predicted cost distributions during the early and final learning stages. The risk-sensitive learning leads to relatively less cost uncertainties due to the penalization on predicted cost variance. Fig. 7 shows

the learning curves for four cases on the same task. We use three and five rollouts at each optimization phase for both risk-neutral and risk-sensitive learning. An interesting phenomenon is that when using only three rollouts (180 data points) for learning, the risk-sensitive policy outperforms risk-neutral policy significantly. This is because when the sample data are sparse in state-action regions, the risk-sensitive criterion restricts exploration in the local region that is close to the sampled data and nominal trajectories. In contrast, in the risk-neutral case, applying PDDP policy results in higher costs, because the cost variance is higher. When using five rollouts (300 data points) per stage, the risk-sensitive policy learns slower than risk-neutral policy in the early stages. But they perform similarly in the final stages. PDDP with both criteria is able to learn the task within about 7–8 optimization stages. Even in the risk-neutral case, PDDP would avoid explorations in highly uncertain regions, because the expected cost (20) depends on the predictive variance of the state. Risk-sensitive learning is more conservative and shows encouraging performance when using a very small amount of training data.

VI. Discussion and Conclusion

In this paper, our goal is to address the principle challenges of applying RL in complex real-world scenarios, namely, data efficiency, computational cost, and scalability. Thus, we have introduced PDDP, a model-based RL framework for systems with unknown or partially known dynamics. PDDP is derived based on two main components: 1) local trajectory optimization via forward–backward sweeps, which is a classical method under the name DDP [9] and 2) probabilistic modeling and approximate inference with GPs. PDDP integrates the aforementioned components by representing the system dynamics using GPs and performing optimization in Gaussian belief spaces. To further improve its applicability, we explored the case of risk-sensitive trajectory optimization.

PDDP generalizes the deterministic [9] and stochastic [10], [11] trajectory optimization to a probabilistic setting, i.e., probabilistic trajectory optimization. By taking advantage of recent development in GPs, PDDP offers data efficiency superior to methods based on learned determinstic models, and comparable to the state-of-the-art GP-based policy search method [22]. Compared with typical gradient-based policy search methods, PDDP features a more computationally efficient policy improvement scheme. PDDP yields local control policies and requires no *a-priori* policy parameterization. These strengths lead to a combination of data efficiency, computational efficiency, and scalability. Theoretically, we provide analyses showing that our algorithm converges to a stationary point globally. To the best of our knowledge, our analyses offer the most general convergence proof for DDP-related methods [9], [10], [12], [15], [16], [26], [39], [40].

The computational efficiency of probabilistic inference can be further improved. The possibility of using sampling-based methods (such as in PEGASUS [56]) instead of deterministic inference has been discussed in [6]. However, it may lead to derivatives with high variance and it is more suitable for sampling-based (gradient-free) methods [58]. Approximate GPs, such as local GP [18], [52] or sparse GP [45], [46], can be exploited to improve the computational efficiency. For instance, sparse spectrum GPs have been used for fast inference in trajectory optimization [55], [57] when: 1) the optimization needs to be performed in real time and receding horizon fashion; 2) training data set is big; and 3) the task, dynamics, and environment are time varying. Application of probabilistic trajectory optimization to real robotic systems is worth for further investigation. In addition, our approximate inference method does not have an explicit error bound. Numerical methods, such as Gaussian quadrature, could provide more accurate approximations and an error bound [33]. However, such numerical methods are usually more computationally expensive than closed-form computation as we do here. Further investigation on combining the benefits of these methods with ours is left for the future work.

Appendix
Prediction via Exact Moment Matching

In this section, we give the expressions of predictive mean and variance using the exact moment matching approach [22], [28], [29]. These expressions are required for computations in Section III-A2. For simplicity, we consider the scalar case when $n = 1$. Given an input joint distribution $p(\tilde{\mathbf{x}}_k) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)$ and apply the law of iterated expectation, we obtain the predictive mean

$$\begin{aligned}\boldsymbol{\mu}_{f_k} &= \int \mathbb{E}_{\mathbf{f}}[\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k]\mathcal{N}(\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)d\tilde{\mathbf{x}}_k \\ &= \underbrace{\left(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_\omega^2\mathbf{I}\right)^{-1}\Delta\mathbf{X}\Big)^T}_{\Psi}\underbrace{\int \mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{x}}_k)\mathcal{N}(\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)d\tilde{\mathbf{x}}_k}_{\mathbf{q}_k}\end{aligned}$$

where $\Psi \in \mathbb{R}^{N\times 1}$ and $\mathbf{q}_k = [q_{k1}, \ldots, q_{kN}]^T \in \mathbb{R}^N$ with each element

$$\begin{aligned}q_{ki} = \sigma_f^2|\tilde{\boldsymbol{\Sigma}}_k\mathbf{W}^{-1} + \mathbf{I}|^{\frac{1}{2}} \\ \times \exp\left(-\frac{1}{2}(\tilde{\mathbf{X}}_i - \tilde{\boldsymbol{\mu}}_k)^T(\tilde{\boldsymbol{\Sigma}}_k + \mathbf{W})^{-1}(\tilde{\mathbf{X}}_i - \tilde{\boldsymbol{\mu}}_k)\right).\end{aligned}$$

Next, we compute the predictive matrix by applying the law of total covariance

$$\begin{aligned}&\boldsymbol{\Sigma}_{f_k} \\ &= \mathbb{E}_{\tilde{\mathbf{x}}_k}[\mathbb{VAR}_{\mathbf{f}}[\mathbf{f}_i(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k]|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] \\ &\quad + \mathbb{E}_{\tilde{\mathbf{x}}_k}[\mathbb{E}_{\mathbf{f}}[\mathbf{f}_i(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k]^2|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] - \boldsymbol{\mu}_{f_{ik}}^2 \\ &= \int \left(\mathbf{K}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_k) - \mathbf{K}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}})\left(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_\omega^2\mathbf{I}\right)^{-1}\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{x}}_k)\right) \\ &\quad \times p(\tilde{\mathbf{x}}_k)d\tilde{\mathbf{x}}_k + \int \left(\mathbf{K}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}})\left(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_\omega^2\mathbf{I}\right)^{-1}\Delta\mathbf{X}\right)^2 \\ &\quad \times p(\tilde{\mathbf{x}}_k)d\tilde{\mathbf{x}}_k - (\Psi^T\mathbf{q}_k)^2 \\ &= \sigma_f^2 - \mathrm{tr}\left(\left(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_\omega^2\mathbf{I}\right)^{-1}\int (\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{x}}_k)\mathbf{K}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}}))p(\tilde{\mathbf{x}}_k)d\tilde{\mathbf{x}}_k\right) \\ &\quad + \Psi^T\underbrace{\left(\int \mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{x}}_k)\mathbf{K}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}})p(\tilde{\mathbf{x}}_k)d\tilde{\mathbf{x}}_k\right)}_{\Phi}\Psi - (\Psi^T\mathbf{q}_k)^2 \\ &= \sigma_f^2 - \mathrm{tr}\left(\left(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_\omega^2\mathbf{I}\right)^{-1}\Phi\right) + \Psi^T\Phi\Psi - (\Psi^T\mathbf{q}_k)^2\end{aligned} \tag{55}$$

where the entries of $\Phi \in \mathbb{R}^{N\times N}$ are given by

$$\Phi_{ij} = \frac{\mathbf{K}(\tilde{\mathbf{X}}_i, \tilde{\boldsymbol{\mu}}_k)\mathbf{K}(\tilde{\mathbf{X}}_j, \tilde{\boldsymbol{\mu}}_k)}{|2\tilde{\boldsymbol{\Sigma}}_k\mathbf{W}^{-1} + I|^{\frac{1}{2}}} \times \exp\left(\left(\frac{1}{2}(\tilde{\mathbf{x}}_i + \tilde{\mathbf{x}}_j) - \tilde{\boldsymbol{\mu}}_k\right)^T \times \left(\tilde{\boldsymbol{\Sigma}} + \frac{1}{2}\mathbf{W}\right)^{-1} \tilde{\boldsymbol{\Sigma}}_k\mathbf{W}^{-1}\left(\frac{1}{2}(\tilde{\mathbf{x}}_i + \tilde{\mathbf{x}}_j) - \tilde{\boldsymbol{\mu}}_k\right)\right).$$

The expressions for the multivariate case can be derived similarly. The derivation is omitted due to the space limit.

REFERENCES

[1] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Netw.*, vol. 21, no. 4, pp. 682–697, May 2008.

[2] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *J. Mach. Learn. Res.*, vol. 11, pp. 3137–3181, Mar. 2010.

[3] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *Proc. AAAI*, 2010, pp. 1–6.

[4] G. Neumann, "Variational inference for policy search in changing situations," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, 2011, pp. 817–824.

[5] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Proc. Int. Conf. Robot. Autom.*, Apr. 1997, pp. 3557–3564.

[6] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Found. Trends Robot.*, vol. 2, nos. 1–2, pp. 1–142, Aug. 2013.

[7] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming* (Optimization and Neural Computation Series, 3), vol. 7. Belmont, MA, USA: Athena Scientific, 1996, pp. 15–23.

[8] A. G. Barto, W. B. Powell, J. Si, and D. Wunsch, Eds., *Handbook of Learning and Approximate Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2004.

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

[10] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proc. Amer. Control Conf.*, Jun. 2005, pp. 300–306.

[11] E. Theodorou, Y. Tassa, and E. Todorov, "Stochastic differential dynamic programming," in *Proc. Amer. Control Conf. (ACC)*, Jun./Jul. 2010, pp. 1125–1132.

[12] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming," in *Proc. NIPS*, 2007, pp. 1465–1472.

[13] J. Morimoto and C. G. Atkeson, "Minimax differential dynamic programming: An application to robust biped walking," in *Proc. NIPS*, 2002, pp. 1539–1546.

[14] Y. Pan, K. Bakshi, and E. Theodorou, "Robust trajectory optimization: A cooperative stochastic game theoretic approach," in *Proc. Robot., Sci. Syst.*, 2015. [Online]. Available: http://www.roboticsproceedings.org/rss11/p29.html

[15] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Proc. NIPS*, vol. 19. 2007, pp. 1–8.

[16] 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*. Berlin, Germany: Springer, 2010, pp. 65–84.

[17] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Proc. NIPS*, vol. 4. 2004, p. 1.

[18] D. Nguyen-Tuong, J. R. Peters, and M. Seeger, "Local Gaussian process regression for real time online model learning," in *Proc. NIPS*, 2008, pp. 1193–1200.

[19] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2010, pp. 2677–2682.

[20] D. Nguyen-Tuong and J. Peters, "Online kernel-based learning for task-space tracking robot control," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 9, pp. 1417–1425, Sep. 2012.

[21] M. Deisenroth and C. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 465–472.

[22] M. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 2, pp. 408–423, Feb. 2015.

[23] P. Hemakumara and S. Sukkarieh, "Learning UAV stability and control derivatives using Gaussian processes," *IEEE Trans. Robot.*, vol. 29, no. 4, pp. 813–824, Aug. 2013.

[24] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela, "Bayesian nonparametric adaptive control using Gaussian processes," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 3, pp. 537–550, Mar. 2015.

[25] P. Dallaire, C. Besse, S. Ross, and B. Chaib-draa, "Bayesian reinforcement learning in continuous POMDPs with Gaussian processes," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2009, pp. 2604–2609.

[26] J. Boedecker, J. T. Springenberg, J. Wülfing, and M. Riedmiller, "Approximate real-time optimal control based on sparse Gaussian process models," in *Proc. IEEE Symp. Adapt. Dyn. Programm. Reinforcement Learn. (ADPRL)*, Dec. 2014, pp. 1–8.

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

[28] J. Q. Candela, A. Girard, J. Larsen, and C. E. Rasmussen, "Propagation of uncertainty in Bayesian kernel models—Application to multiple-step ahead forecasting," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Apr. 2003, pp. II-701–II-704.

[29] A. Girard, C. Rasmussen, J. Q. Candela, and R. Murray-Smith, "Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2003, pp. 545–552.

[30] M. Kuss, "Gaussian process models for robust regression, classification, and reinforcement learning," Ph.D. dissertation, Dept. Comput. Sci., Tech. Univ. Munich, Munich, Germany, 2006.

[31] M. P. Deisenroth, M. F. Huber, and U. D. Hanebeck, "Analytic moment-based Gaussian process filtering," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 225–232.

[32] Y. Pan, E. Theodorou, and M. Kontitsis, "Sample efficient path integral control under uncertainty," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2314–2322.

[33] J. Vinogradska, B. Bischoff, D. Nguyen-Tuong, H. Schmidt, A. Romer, and J. Peters, "Stability of controllers for Gaussian process forward models," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 545–554.

[34] A. O'Hagan and J. F. C. Kingman, "Curve fitting and optimal design for prediction," *J. Roy. Statist. Soc. B, Methodol.*, vol. 40, no. 1, pp. 1–42, 1978.

[35] T. Wu and J. Movellan, "Semi-parametric Gaussian process for robot system identification," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2012, pp. 725–731.

[36] E. M. Schearer *et al.*, "Identifying inverse human arm dynamics using a robotic testbed," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2014, pp. 3585–3591.

[37] Z. I. Botev *et al.*, "The cross-entropy method for optimization," *Machine Learning: Theory and Applications*, vol. 31, V. Govindaraju and C. R. Rao, Eds. Chennai, India: Elsevier, 2013, pp. 35–59.

[38] P. Whittle, *Risk-Sensitive Optimal Control*. Hoboken, NJ, USA: Wiley, 1990.

[39] L.-Z. Liao and C. A. Shoemaker, "Convergence in unconstrained discrete-time differential dynamic programming," *IEEE Trans. Autom. Control*, vol. 36, no. 6, pp. 692–706, Jun. 1991.

[40] L.-Z. Liao, "Global convergence of differential dynamic programming and Newton's method for discrete-time optimal control," Dept. Math., Hong Kong Baptist Univ., Hong Kong, Tech. Rep., 1996.

[41] G. I. Boutselis, Y. Pan, G. De La Tore, and E. A. Theodorou. (2017). "Stochastic trajectory optimization for mechanical systems with parametric uncertainties." [Online]. Available: https://arxiv.org/abs/1705.05506

[42] J. van den Berg, S. Patil, and R. Alterovitz, "Motion planning under uncertainty using iterative local optimization in belief space," *Int. J. Robot. Res.*, vol. 31, no. 11, pp. 1263–1278, 2012.

[43] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Proc. ICRA*, May/Jun. 2014, pp. 1168–1175.

[44] D. P. Bertsekas, "Projected Newton methods for optimization problems with simple constraints," *SIAM J. Control Optim.*, vol. 20, no. 2, pp. 221–246, 1982.

[45] L. Csató and M. Opper, "Sparse on-line Gaussian processes," *Neural Comput.*, vol. 14, no. 3, pp. 641–668, Mar. 2002.

[46] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 18. 2006, pp. 1257–1264.
[47] Y. Pan and E. Theodorou, "Probabilistic differential dynamic programming," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 1907–1915.
[48] T. N. Hoang, K. H. Low, P. Jaillet, and M. Kankanhalli, "Nonmyopic ϵ-Bayes-optimal active learning of Gaussian processes," in *Proc. ICML*, 2014, pp. 739–747.
[49] C. K. Ling, K. H. Low, and P. Jaillet, "Gaussian process planning with Lipschitz continuous reward functions: Towards unifying Bayesian optimization, active learning, and beyond," in *Proc. AAAI*, 2016, pp. 1–7.
[50] J. van den Berg, S. Patil, and R. Alterovitz, "Motion planning under uncertainty using differential dynamic programming in belief space," in *Proc. Int. Symp. Robot. Res.*, 2011, pp. 473–490.
[51] J. Morimoto, G. Zeglin, and C. G. Atkeson, "Minimax differential dynamic programming: Application to a biped walking robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, vol. 2. Oct. 2003, pp. 1927–1932.
[52] F. Meier, P. Hennig, and S. Schaal, "Incremental local Gaussian regression," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 972–980.
[53] M. Lázaro-Gredilla, J. Quiñonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal, "Sparse spectrum Gaussian process regression," *J. Mach. Learn. Res.*, vol. 11, pp. 1865–1881, Mar. 2010.
[54] Y. Pan, X. Yan, E. Theodorou, and B. Boots. (2016). "Adaptive probabilistic trajectory optimization via efficient approximate inference." [Online]. Available: https://arxiv.org/abs/1608.06235
[55] Y. Pan, X. Yan, E. A. Theodorou, and B. Boots, "Prediction under uncertainty in sparse spectrum Gaussian processes with applications to filtering and control," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2760–2768.
[56] A. Y. Ng and M. Jordan, "PEGASUS: A policy search method for large MDPs and POMDPs," in *Proc. 16th Conf. Uncertainty Artif. Intell.*, 2000, pp. 406–415.
[57] M. Gandhi, Y. Pan, and E. Theodorou. (2017). "Pseudospectral model predictive control under partially learned dynamics." [Online]. Available: https://arxiv.org/abs/1702.04800
[58] Y. Pan, K. Saigol, and E. A. Theodorou, "Belief space stochastic control under unknown dynamics," in *Proc. Amer. Control Conf. (ACC)*, May 2017, pp. 3764–3770.

Yunpeng Pan received the B.S. degree in automation from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2007, the M.Phil. degree in mechanical and automation engineering from The Chinese University of Hong Kong, Hong Kong, in 2009, and the M.S. degree in aerospace engineering and the Ph.D. degree in robotics from the Georgia Institute of Technology, Atlanta, GA, USA, in 2014 and 2017, respectively.

He is a Senior Scientist at JD.com's JD-X Silicon Valley Research Center, Santa Clara, CA, USA. His current research interests include statistical machine learning, reinforcement learning, stochastic optimal control, and their applications to robotics.

George I. Boutselis received the Diploma degree in mechanical engineering from the National Technical University of Athens, Athens, Greece, in 2014, and the M.S. degree in aerospace engineering from the Georgia Institute of Technology (Georgia Tech), Atlanta, GA, USA, in 2016, where he is currently pursuing the Ph.D. degree.

He is a member of the Autonomous Control and Decision Systems Laboratory, Georgia Tech. His current research interests include numerical optimal control, robotics, geometric control, and machine learning.

Evangelos A. Theodorou received the Diploma degree in electronic and computer engineering and the M.Sc. degree in production engineering from the Technical University of Crete, Chania, Greece, in 2001 and 2003, respectively, the M.Sc. degree in computer science and engineering from the University of Minnesota, Minneapolis, MN, USA, in 2007, and the M.Sc. degree in electrical engineering on dynamics and controls and the Ph.D. degree in computer science from the University of Southern California, Los Angeles, CA, USA, in 2010 and 2011, respectively.

In 2011, he became a Post-Doctoral Research Associate with the Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA. He is affiliated with the Institute of Robotics and Intelligent Machines and the Center for Machine Learning, Georgia Institute of Technology (Georgia Tech), Atlanta, GA, USA. He is currently an Assistant Professor with the Guggenheim School of Aerospace Engineering, Georgia Tech. His current research interests include the areas of stochastic optimal control theory, machine learning, information theory, and statistical physics.

Dr. Theodorou was a recipient of the King-Sun Fu Best Paper Award of the IEEE TRANSACTIONS ON ROBOTICS in 2012 and the Best Paper Award in Cognitive Robotics from the International Conference of Robotics and Automation 2011. He was a finalist for the Best Paper Award at the International Conference of Humanoid Robotics 2010 and the International Conference of Robotics and Automation 2017.