Gaussian Processes for Learning and Control

A TUTORIAL WITH EXAMPLES

MIAO LIU, GIRISH CHOWDHARY, BRUNO CASTRO DA SILVA, SHIH-YUAN LIU, and JONATHAN P. HOW

Many challenging real-world control problems require adaptation and learning in the presence of uncertainty. Examples of these challenging domains include aircraft adaptive control under uncertain disturbances [1], [2], multiple-vehicle tracking with space-dependent uncertain dynamics [3], [4], robotic-arm control [5], blimp control [6], [7], mobile robot tracking and localization [8], [9], cart-pole systems and unicycle control [10], gait optimization in legged robots [11] and snake robots [12], and any other system whose dynamics are uncertain and for which limited data are available for model learning. Classical model reference adaptive control (MRAC) [13]–[15] and reinforcement

Digital Object Identifier 10.1109/MCS.2018.2851010
Date of publication: 18 September 2018

Summary

Gaussian processes (GPs) are data-driven machine learning models that have been widely used in tasks such as regression (modeling the behavior of an unknown system or function based on samples) and classification (for example, in pattern recognition). GP methods applied to control tasks still remain (with only a few notable exceptions) largely unexploited by the general controls community. This tutorial discusses key features of GPs: 1) how they allow for automatic, data-driven feature selection, thereby not requiring practitioners to predefine the number and nature of the features that describe the system being modeled, 2) the way in which GPs offer uncertainty measures over predictions, and 3) how GPs offer a principled way to perform budgeted online inference. This tutorial also discusses a few limitations of the basic GP-based approaches and provides a brief overview of several advanced GP models that overcome such limitations. Finally, the tutorial provides concrete examples demonstrating how GPs can be used for adaptive control and off-policy reinforcement learning and to model value functions in optimal control and planning problems and reward functions in inverse optimal control problems.

learning (RL) methods [16]–[23] have been developed to address these challenges and rely on parametric adaptive elements or control policies whose number of parameters or features are fixed and determined a priori. One example of such an adaptive model are radial basis function networks (RBFNs), with RBF centers pre-allocated based on expected operating domains [24], [25].

As discussed in "Summary," a potential problem with this approach is that, if the system operates outside of the expected domain, then adaptive elements may not correctly model uncertainty. The defect in uncertainty modeling may lead to system instability, thereby rendering any controller stability results only semiglobal in nature. "Limitations of Parametric Models—A Flight Control Example" illustrates this issue through an adaptive flight control application using parametric neural networks (NNs).

Nonparametric models have been designed because methods with fixed parametric structure lack the robustness needed to capture uncertainty outside their expected operating domains. In a nonparametric model, the number of parameters and their properties are not fixed but rather grow and adjust with the amount of training data. Within the class of nonparametric modeling methods, *Bayesian modeling* leads to data-driven generative models optimized to fit data under explicit assumptions of measurement uncertainty. Gaussian processes (GPs) [26] are highlighted as an important class of Bayesian nonparametric models (BNPMs). GPs are priors over a class of functions or models of interest, for instance, models describing the dynamics of an unknown system. GPs can directly encode probability distributions over such functions or models, given observed data, and allow for updated posterior distributions over models or functions given additional sampled data. They also allow for data-driven feature construction and selection and do not require a designer to explicitly predefine feature numbers and locations. Finally, they explicitly incorporate and provide uncertainty estimates in all of their predictions.

The basic theory underpinning GPs can be traced back to the work of Wiener [27] and Kolmogorov [28] on time-series analysis. GP regression (GPR), also named Kriging in spatial statistics [29], [30], is a widely used method that was developed in the 1970s. Because of the high computational complexity of GP-based methods, however, they did not become widely used in the machine learning and engineering communities until the early 21st century. Since then, numerous articles on GPs have been published in major conference proceedings and journals on machine learning and robotics. GPs are also deeply connected with kernel-filtering methods, made possible by the reproducing kernel Hilbert space (RKHS) interpretation of GPR [31]. However, using GPR for function approximation yields additional benefits over kernel-filtering methods: no prior knowledge of the uncertainty over operating domains is required (but can be used if available), measurement noise is handled inherently, and features can be automatically inferred without manual specification.

As previously mentioned, GPs encode uncertainty over a domain as *distributions over functions*, which differs from the traditional deterministic, weight-space-based approaches, where uncertainty is characterized by a model's error function [32]–[35]. In addition, GPR is heavily based on Bayesian inference, and when the likelihood is Gaussian (that is, when Gaussian RBFs are used), GP inference is analytic and can be solved with a least squares formulation. This avoids the necessity to utilize gradient-based update laws, which can suffer from a lack of convergence due to being stuck in local minima. Furthermore, in the context of MRAC, GP updates lead to guaranteed parameter convergence [36] when the data are exciting over an interval and are not susceptible to parameter bursting (parameters growing without bound [37]–[39] in the presence of noise), since parameter regularization can be utilized in the learning scheme itself.

Due to their representation flexibility and the advancement of tractable learning algorithms, GP models have been widely used in machine-learning tasks [26], [40] such as regression, classification, and dimensionality reduction. However, GP-related, data-driven control methods remain, with notable exceptions [10], [41]–[46], largely unexploited by the general controls community. This tutorial presents a self-contained introduction to GPR techniques and discusses several examples of their use in solving control-related

Limitations of Parametric Models—A Flight Control Example

Model reference adaptive control (MRAC) is a widely studied adaptive control methodology for aerospace applications that aims to ensure stability and performance in the presence of model uncertainty. A central element of many MRAC architectures is a parameterized adaptive element, whose parameters are tuned online by the MRAC architecture to capture the uncertainty of a model. A common approach to allow for the parameters of the adaptive element to be tuned online is to represent it as a deterministic weighted combination of a set of basis functions. This approach has the benefit of allowing a designer to directly encode any physical knowledge about the modeling uncertainty into the parameters. However, in many aerospace or mechanical problems of interest, the exact form of the adaptive element may not be known a priori, especially when the underlying uncertainties result from highly stochastic effects, such as unsteady aerodynamics, or nonlinear dynamics, such as actuator limitations.

A set of flight control experiments was performed in the Aerospace Control Laboratory's Real-Time Indoor Autonomous Vehicle Test Environment (RAVEN) test environment at the Massachusetts Institute of Technology [76] to evaluate the performance of different MRAC strategies. A quadrotor was tasked with performing "figure-8" maneuvers. Five trials consisting of 20 maneuvers per trial were performed with a period of 6.28 s, taking approximately 125 s per trial. Figure 7(a) and (b) depict the trajectories in space, as followed by the vehicle. When using a baseline proportional-integral-derivative controller, the quadrotor was unable to track the figure-8 trajectory. Concurrent learning (CL)-MRAC performed better at this task [80] (although it overshot the trajectory at points), resulting in sharp corners. The reason for this inability to follow the desired trajectory is that the adaptive elements in CL-MRAC are modeled via a parametric model with pre-allocated and uniformly spaced feature vectors, which do not accurately capture all the features of the dynamics along the trajectory resulting from the figure-8 shape.

The difficulty of CL-MRAC in tackling this problem results from the fact that it is a *parametric method*: the form of the adaptive elements is fixed a priori. Other parametric methods have been used to construct adaptive controllers, but most of them share similar limitations. Neural networks (NNs) and radial basic function networks (RBFNs), for instance, have been used in the control and estimation literature as models of uncertain dynamics and measurement equations, whenever physics-based models are unavailable [56], [65], [71], [S1], [S2]. Most NN models (including single hidden-layer NNs) are nonlinearly parameterized [56], [71], [S1]–[S3], which causes commonly used learning methods (such as backpropagation) to get stuck in local optima [32]. RBFNs, on the other hand, are linearly parameterized NNs where the output is computed by linearly combining a set of exponential RBFs, all of which have fixed and predetermined centers in the input space. Let $\mathbf{z}$ denote an input variable to an RBFN. The output of the RBFN is

$$g(\mathbf{z}) = \hat{\mathbf{w}}^T \phi(\mathbf{z}), \tag{S1}$$

where $\hat{\mathbf{w}} \in \mathbb{R}^l$ and $\phi(\mathbf{z}) = [1, \phi_2(\mathbf{z}), \phi_3(\mathbf{z}), \ldots, \phi_l(\mathbf{z})]^T$ is a vector of known Gaussian RBFs. For $i = 2, 3..., l$, let $\mathbf{c}_i$ and μ_i denote the centroid and width, respectively, of the ith RBF. Each RBF can thus be expressed as

$$\phi_i(\mathbf{z}) = e^{-\|\mathbf{z}-\mathbf{c}_i\|^2/\mu_i}. \tag{S2}$$

RBFNs are parametric models that have gained popularity in adaptive control research since global optimal solutions can, in theory, be found by using gradient descent techniques. In practice, however, RBFNs also present problems: it may be difficult to determine how many RBFs should be used and where their centers should lie. RBFNs rely on the universal approximation property of RBFN [66], which asserts that given a *fixed* number of RBFs l, there exists a set of optimal weights $\mathbf{w}^* \in \mathbb{R}^l$ and a real number $\tilde{\epsilon}$ such that

$$g(\mathbf{z}) = \mathbf{w}^{*T} \phi(\mathbf{z}) + \tilde{\epsilon}, \tag{S3}$$

where $\tilde{\epsilon}$ can be made arbitrarily small given a sufficiently large number of RBFs. Even though this parametric model comes with strong formal guarantees about its approximation capability, when deploying RBFNs, convergence of weights to their true optimal values depends on the amount of excitation "visible" to the learning law. It is shown in [33] that the location of RBF centers directly affects the amount of excitation that is visible to the adaptive law in an MRAC setting. That is, even if $\mathbf{z}(t)$ is exciting, when the system evolves away from where the centers are distributed, persistent excitation of $\phi(\mathbf{z})$ may fail to hold. In other words, $\mathbf{z}$ may never (in practice) be close to where the RBFs are centered, and hence (S1) will be very close to zero, regardless of the weights.

REFERENCES

[S1] F. L. Lewis, "Nonlinear network structures for feedback control," *Asian J. Control*, vol. 1, no. 4, pp. 205–228, Dec. 1999.

[S2] A. J. Calise and R. T. Rysdyk, "Nonlinear adaptive flight control using neural networks," *IEEE Control Syst. Mag.*, vol. 18, no. 6, pp. 14–25, Dec. 1998.

[S3] S. Kannan, "Adaptive control of systems in cascade with saturation," Ph.D. dissertation, Georgia Inst. Technol., Atlanta, GA, 2005.

problems, such as optimal adaptive control, estimation and filtering, planning, RL, and inverse optimal control (IRL). Other important aspects of GP models (such as the efficient computation of GP posteriors based on sparse approximations and connections to other machine-learning models) are also discussed in detail.

Although many textbooks and tutorials on GPs exist [26], [47]–[50], most are directed to the machine-learning community and focus on data-mining applications. To spark the control community's interest in GP-related techniques, this tutorial uses control and RL applications as motivating examples and shows how GPs (when used as universal function approximators) can model key functions underpinning many control problems, including system dynamics, measurement models, value functions, control policies, and cost or reward functions (Figure 1). The insights and lessons from the applications discussed here are expected to provide a guide to practitioners so that they can recognize ways in which to apply GPR and modeling to their own control and RL problems of interest.

This tutorial is organized as follows. The next section provides an overview of the basics of GPR, including generative models and Bayesian inference. Next, applications of GPR are presented with examples from different control subfields, including MRAC, planning, RL, and IRL. In addition, this tutorial also discusses limitations of the basic GP model and introduces recent extensions. Finally, the tutorial presents online resources and software, in tandem with popular implementations that can help practitioners to use GPs to solve particular problems.

The remainder of this tutorial adopts the following notations: boldface upper-case letters denote matrices ($\boldsymbol{X}$) and boldface lower-case letters denote column vectors ($\boldsymbol{x}$). The symbols $\boldsymbol{I}_n$, $\boldsymbol{0}_{m\times n}$, and 1_n represent the identity matrix of dimension $n\times n$, the all-zero-entries matrix of dimension $m\times n$, and the all-one-entires vectors of dimension n, respectively (subscripts will be dropped when the dimensions are clear from the context). Nonboldface, upper-case, calligraphic letters ($\mathcal{X}$) denote sets, and nonboldface and nonscript letters denote scalars (x, X).

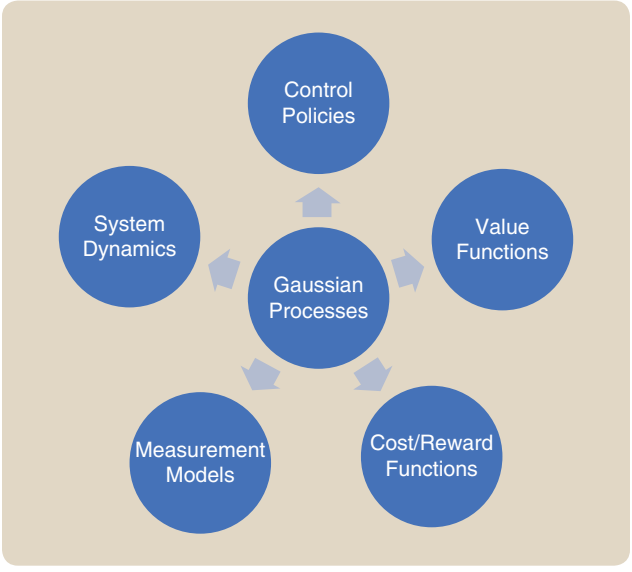

FIGURE 1 Gaussian processes are universal function approximators that can be used for modeling various functions relevant to optimal control and decision making.

INTRODUCTION TO GAUSSIAN PROCESSES

Generative Models and Bayesian Inference for Gaussian Processes

There exists several possible ways to mathematically motivate the theory underlying GPs, including views from weight space, function space, and Bayesian nonparametrics. The approach in this tutorial is the Bayesian nonparametric perspective since it offers an intuitive and principled way for performing inference under uncertainty. Other mathematical formulations are also useful and offer different views to understand GPs; their details can be found in [26] and [48].

It is instructive at this point to define what is meant by generative models. A model is termed generative [26], [48], [49], [51] when it can randomly generate the data it learned from its distribution. That is, after a generative model is trained, the joint probability distribution of the data generated by a generative model's prediction is the same as the distribution of the data it was trained on. GPs are generative models on distributions over functions, that is, a random draw from a GP is a function, and many such random draws have the same distribution over functions as the training samples obtained from the functions on which they were trained. In essence, when the noise is Gaussian, GPs predict the mean of the function they attempt to model.

Gaussian Processes Priors

GPs are stochastic processes that can be used to model continuous functions. They are infinite-dimensional extensions to multivariate Gaussian distributions; specifically, they are a (possibly infinite) set of random variables, any finite set of which is jointly Gaussian distributed. Since GPs extend Gaussian distributions to the case of infinite dimensionality, they can be seen and used as distributions over functions. The expressiveness and flexibility of GPs make them ideal choices as nonlinear function approximators whenever it is difficult to specify parametric forms for unknown functions and models, for example, the model of uncertain wind disturbance in aircraft controls [1], [2].

Formally, a GP is a collection of infinite continuous random variables $\mathcal{G}$, any finite subset $\{g(z_1),\ldots g(z_\tau)\}\subset\mathcal{G}$ of which has a joint Gaussian distribution. Here, $\{z_1,\ldots z_\tau\}\subset\mathcal{Z}$

corresponds to the space of inputs of a function or model to be estimated or represented nonparametrically. In general, each z_i is a d-dimensional point in Euclidean space $\mathbb{R}^d$. A function $g:\mathbb{R}^d \to \mathbb{R}$ is a sample drawn from a GP denoted as

$$g(z) \sim \mathcal{GP}(m(z), k(z,z')) \tag{1}$$

when the probability density of g at any location $z \in \mathcal{Z}$ is jointly normal. A GP is characterized by a mean function

$$m(z) = \mathbb{E}[g(z)], \tag{2}$$

and a covariance function

$$k(z,z') = \mathbb{E}[(g(z) - m(z))(g(z') - m(z'))]. \tag{3}$$

The covariance function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ maps two vector inputs to a scalar output and is usually specified by a positive-definite kernel $k(z,z')$. Kernel functions capture the similarity between inputs of a function or model and play a pivotal role in GP predictions. Intuitively, a kernel specifies properties of the function being modeled or estimated through a GP, such as smoothness and periodicity. If it is known, for instance, that the function being estimated is smooth—its outputs vary slowly as its inputs change—it is possible to encode that assumption in the kernel function used by the GP. Because $k(z,z')$ captures the similarity between inputs z and z', larger values for $k(z,z')$ imply that the corresponding outputs $g(z)$ and $g(z')$ of the function are highly correlated—the outputs tend to be similar for both inputs. Therefore, intermediate input points between z and z' also have corresponding outputs that are highly correlated with $g(z)$ and $g(z')$: the function is smooth in that interval.

Note that, according to (3), kernel functions are always symmetric and positive. Based on how they are defined, kernel functions can be classified into different categories. A *stationary kernel* is a function of $r = |z - z'|$ and is shift invariant in the input space. Anisotropic versions of these isotropic kernel functions can be constructed by setting $r^2(z,z') = (z-z')^T \boldsymbol{M}(z-z')$ for some positive-semidefinite matrix $\boldsymbol{M}$. Different kernels are useful in different contexts; for example, the *polynomial kernel* is often used in high-dimensional classification problems. Several commonly used kernel functions are summarized in Table 1. One commonly used kernel functions is the square exponential covariance function (also called the RBF kernel) defined as

$$k(z,z') = A\exp\left\{-\sum_{i=1}^{d}\frac{(z[i]-z'[i])^2}{2\sigma_i^2}\right\} + \omega_n^2\delta(z,z'), \tag{4}$$

where A is an amplitude hyperparameter, σ_i is a hyperparameter describing a scaling factor along dimension i of the input space, and ω_n^2 is a hyperparameter encoding the variance of measurement noise. The ratio between σ_i and ω_n characterizes the relative effects of the measurement noise and the influence of nearby observations when predicting the value of a function or model at a particular input location. To better understand the roles of these hyperparameters, Figure 2 illustrates the effect of varying their values. In Figure 2, comparing (a) with (b) (given the same values for A and ω_n), it is shown that the GP with smaller σ_i has more oscillations since reducing the scaling factor mitigates possible correlations between nearby points. Comparing Figure 2(b) with (c) (given the same values for A and σ_i), it is possible to note that the GP with higher noise variance ω_n^2 results in a model with more uncertainty, as indicated by a wider shaded region. Comparing Figure 2(a) and (d), given the same values for σ_i and ω_n, the GP with smaller amplitude (A) results in a model that captures lower correlations between inputs that are far apart.

These plots show that the hyperparameters have a significant impact on the shape of regression functions; hence they must be carefully selected to optimize certain performance measures. More details regarding the automatic inference of hyperparameters is discussed in the next section.

Gaussian Process Regression

As previously mentioned, GPs encode prior distributions over functions or models and can be updated to form posterior distributions given new observed data. A natural application of GPs, then, is in regression tasks, where estimates of an underlying function or model given observed input–output

TABLE 1 A summary of several commonly used kernel functions. The kernels are written either as a function of z and z' or as a function of $\delta = |z - z'|$, where $z, z' \in \mathbb{R}^d$. The dimensionality of input vector is d. $K_\nu(\cdot)$ is a modified Bessel function (see [26, Sec. 4.2] for more details).

Kernel Functions	Expression for $k(z, z')$	Stationary
Constant	σ_0	Yes
Linear	$\sum_{i=1}^{d}\sigma_i^2 z(i) z(i)'$	No
Polynomial	$(\sigma_0^2 + \boldsymbol{z}^T\Sigma_p\boldsymbol{z}')^p$	No
Square exponential	$\exp\left(-\frac{\delta^2}{2l^2}\right)$	Yes
Matérn	$\frac{1}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{l}\right)^{\nu} K_\nu\left(\frac{\sqrt{(2\nu)}}{l}\right)$	Yes
γ-exponential	$k(\delta) = \exp\left(-\left(\frac{\delta}{l}\right)^{\gamma}\right)$ for $0 < \gamma \leq 2$	Yes
Rational quadratic (RQ)	$k_{RQ}(\delta) = \left(1 + \frac{\delta^2}{2\alpha^2}\right)^{-\alpha}$	Yes
Neural network	$\sin^{-1}\left(\frac{2\tilde{\boldsymbol{z}}^T\Sigma\tilde{\boldsymbol{z}}'}{\sqrt{(1+2\tilde{\boldsymbol{z}}^T\Sigma\tilde{\boldsymbol{z}})(1+2\tilde{\boldsymbol{z}}'^T\Sigma\tilde{\boldsymbol{z}}')}}\right)$	No

samples are needed. This type of application is known as GPR. The following sections describe how GP priors can be used both to define distributions over functions and derive a regression algorithm from a Bayesian perspective, which offers a principled method for handling uncertainty.

Gaussian Processes as Distributions over Functions

This section discusses how GPs can be used to define distributions over functions; these results will be used afterward to derive a GPR algorithm. Assume a finite set of input locations $\mathcal{Z}_\tau = \{z_1, \ldots, z_\tau\}$ and corresponding measured outputs $\boldsymbol{g}_\tau = [g(z_1), \ldots, g(z_\tau)]^T$ of some unknown function g. A GP prior encodes a distribution $p(\boldsymbol{g}_\tau | \mathcal{Z}_\tau)$ over the underlying functions that may have generated such observations. Specifically, $p(\boldsymbol{g}_\tau | \mathcal{Z}_\tau)$ is a multidimensional Gaussian defined by its mean vector $\boldsymbol{\mu}_\tau$ and covariance matrix $\boldsymbol{K}_\tau$,

$$p(\boldsymbol{g}_\tau | \mathcal{Z}_\tau) = \mathcal{N}(\boldsymbol{\mu}_\tau, \boldsymbol{K}_\tau). \tag{5}$$

In a real system, access to $\boldsymbol{g}_\tau$ (the noise-free observations of an underlying unknown function or model) is difficult, since they are hidden variables. Instead, noisy samples $\boldsymbol{y}_\tau = [y_1, \ldots, y_\tau]^T$ of g (obtained from a measurement model) are typically accessible. A commonly used measurement model can be formulated as

$$y = g(z) + \epsilon, \tag{6}$$

where $\epsilon \sim \mathcal{N}(0, \omega_n^2)$ is the measurement noise with zero mean and variance ω_n^2.

Given the prior distribution over outputs (5) and a measurement model (6), the posterior distribution over g can be computed, and a distribution over the possible values of g

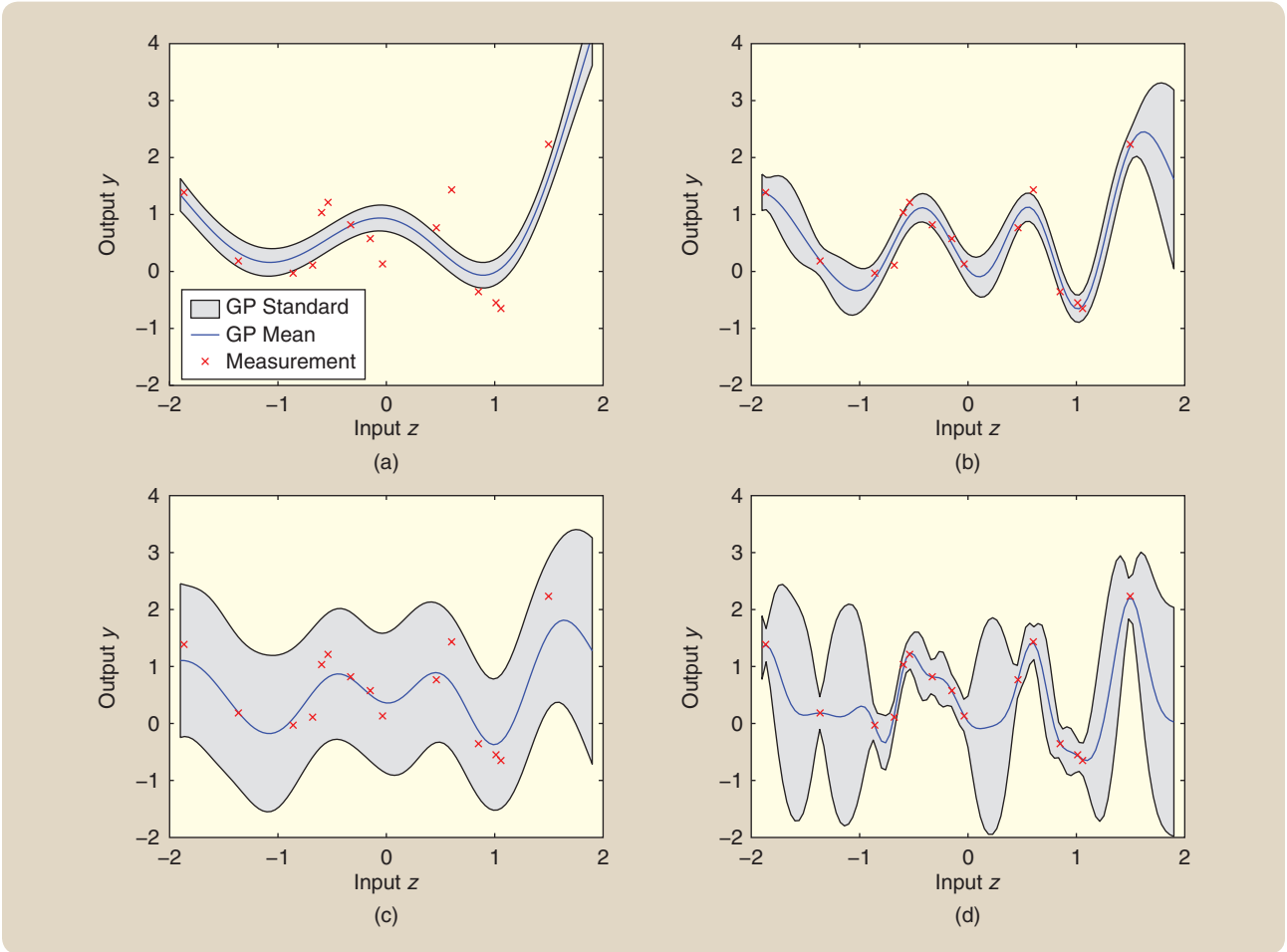

FIGURE 2 An illustration of one-dimensional Gaussian processes (GPs) fitted with radial basis function kernels with different hyperparameters. The hyperparameters (A, σ_i, w_n) are set to (a) (1, 1, 0.1), (b) (1, 0.3679, 0.1), (c) (1,0.3679,0.5), and (d) (0.1353, 1, 0.1), respectively. Reds crosses represent measurements. Comparing (a) with (b), given the same values for A and w_n, it is shown that the GP with smaller σ_i has more oscillations since reducing the scaling factor mitigates possible correlations between nearby points. Comparing (b) with (c), given the same values for A and σ_i, it is possible to note that the GP with higher noise variance w_n^2 results in a model with more uncertainty, as indicated by a wider shaded region. Comparing (a) and (d), given same values for σ_i and w_n, note that the GP with smaller amplitude A results in a model that captures lower correlations between inputs that are far apart.

at a new location $z_{\tau+1}$ can be derived before a sample at that location is observed. To this end, first let $\boldsymbol{K}(\mathcal{Z}_\tau, \mathcal{Z}_\tau) \in \mathbb{R}^{\tau\times\tau}$ be a kernel matrix with entries $\boldsymbol{K}_{i,j} = k(z_i, z_j)$ and $\mathbf{k}(\mathcal{Z}_\tau, z_{\tau+1}) \in \mathbb{R}^\tau$ be a kernel vector with entries $k(z_i, z_{\tau+1})$ for all $i \in 1, \ldots, \tau+1$ associated with the $\tau+1$th measurement. Applying Bayes's rule, the posterior distribution of the latent function g at locations $\mathcal{Z}_{\tau+1} = \mathcal{Z}_\tau \cup \{z_{\tau+1}\}$ (which includes the unobserved location $z_{\tau+1}$ where a prediction is made) can be obtained as follows:

$$p(\boldsymbol{g}_{\tau+1}|\,\boldsymbol{y}_\tau, \mathcal{Z}_{\tau+1}) = \frac{p(\boldsymbol{y}_\tau|\,\boldsymbol{g}_\tau)p(\boldsymbol{g}_{\tau+1}|\,\mathcal{Z}_{\tau+1})}{p(\boldsymbol{y}_\tau|\,\mathcal{Z}_\tau)}, \tag{7}$$

where $p(\boldsymbol{g}_{\tau+1}|\,\mathcal{Z}_{\tau+1})$ is the GP prior extending (5) with a new input $z_{\tau+1}$, and

$$p(\boldsymbol{y}_\tau|\,\boldsymbol{g}_\tau) = \prod_{i=1}^{\tau} p(y_i|\,g(z_i)) = \prod_{i=1}^{\tau} \mathcal{N}(g(z_i), \omega_n), \tag{8}$$

is the likelihood of the latent function g with Gaussian noise assumption. Note that the measurement noise ϵ is not required to be Gaussian; the Gaussian assumption is convenient because it results in closed-form posteriors and for this reason is used often.

A Gaussian Process-Based Regression Algorithm

The Bayesian interpretation introduced in the previous section defines a posterior distribution over unobserved locations of some function g. This section makes use of these results to derive a GPR algorithm. The first step toward achieving this goal involves marginalizing $\boldsymbol{g}_\tau$ in (7). The posterior predictive density of the output g at a new input location $z_{\tau+1}$ [conditioned on training data $(\mathcal{Z}_\tau, y_\tau)$] can then be derived as

$$\begin{aligned} p(g(z_{\tau+1})|\,\mathcal{Z}_\tau, \boldsymbol{y}_\tau, z_{\tau+1}) &= \int p(\boldsymbol{g}_{\tau+1}|\,\boldsymbol{y}_\tau, \mathcal{Z}_{\tau+1}) d\boldsymbol{g}_\tau, \\ &= \mathcal{N}(m(z_{\tau+1}), \Sigma(z_{\tau+1})), \end{aligned} \tag{9}$$

where

$$m(z_{\tau+1}) = \mu(z_{\tau+1}) + \boldsymbol{\alpha}_\tau^T \mathbf{k}(\mathcal{Z}_\tau, z_{\tau+1}) \tag{10}$$

is the predictive mean of the output at location $z_{\tau+1}$, with

$$\boldsymbol{\alpha}_\tau = [\boldsymbol{K}(\mathcal{Z}_\tau, \mathcal{Z}_\tau) + \omega_n^2 \boldsymbol{I}_\tau]^{-1}(\boldsymbol{y} - \mu(\mathcal{Z}_\tau)) \tag{11}$$

being the kernel weights and

$$\begin{aligned} \sum(z_{\tau+1}) = k(z_{\tau+1}, z_{\tau+1}) + \omega_n^2 - \mathbf{k}^T(\mathcal{Z}_\tau, z_{\tau+1}) \\ \times [\boldsymbol{K}(\mathcal{Z}_\tau, \mathcal{Z}_\tau) + \omega_n^2 \boldsymbol{I}_\tau]^{-1} \mathbf{k}(\mathcal{Z}_\tau, z_{\tau+1}), \end{aligned} \tag{12}$$

the predictive covariance. In other words, (10) and (12) consist of GP-based mathematical models for predicting the mean value (and corresponding covariance) of any unknown location of some latent function g. They can, therefore, be used to construct a GPR algorithm that predicts the value of some latent function g at any locations of interest. First, note that (10) is the solution of the kernel ridge regression (KRR) [48]. This implies that GPR computes the mean predicted output at location $z_{\tau+1}$ by linearly combining a vector of kernel functions $\mathbf{k}(\mathcal{Z}_\tau, z_{\tau+1})$ and a feature vector of kernel weights $\boldsymbol{\alpha}_\tau$. Even though (10) is equivalent to the solution of KRR, GPR differs from KRR by providing additional uncertainty measures over the possible outputs of g at any locations z_i, specifically, the predictive variance as given by (12).

Before the GPR algorithm is introduced, it is useful to present a pictorial example of the posterior inference procedure performed by it. Consider the GPs prior and posterior models based on two input locations z_1 and z_2, as shown in Figure 3. Before making any measurements of g at z_1 and z_2, the joint prior distribution of outputs at these locations is a two-dimensional Gaussian with mean vector $[m_1, m_2]^T$ and covariance matrix

$$E[(g(z_1) - m_1)(g(z_2) - m_2)] = \begin{bmatrix} k(z_1, z_1) & k(z_1, z_2) \\ k(z_1, z_2) & k(z_2, z_2) \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} k_0 & k_{1,2} \\ k_{2,1} & k_0 \end{bmatrix}.$$

Such a joint prior is illustrated in Figure 3(a). Figure 3(b) shows the marginal prior distribution of the outputs at locations z_1 and z_2. Figure 3(c) visualizes the joint *posterior* of $g(z_1)$ and $g(z_2)$, after a measurement, y [the noisy version of $g(z_1)$] has been obtained at location z_1. Accordingly, by applying (10) and (12), the conditional marginal posterior of outputs at locations z_1 and z_2 can be obtained as

$$\begin{aligned} p(g(z_1)|\,m_1, z_1, y) &= \mathcal{N}\left(\frac{k_0 y + w_n^2 m_1}{k_0 + w_n^2}, \frac{k_0 w_n^2}{k_0 + w_n^2}\right), \\ p(g(z_2)|\,m_2, z_2, y) &= \mathcal{N}\Big(m_2 + \frac{k_{1,2}}{k_0 + w_n^2}(y - m_1), \\ &\qquad k_0 + w_n^2 - \frac{k_{1,2}^2}{k_0 + w_n^2}\Big). \end{aligned} \tag{13}$$

The distributions in (13) are visualized in Figure 3(d).

Given the results described above, it is easy to devise an algorithm that implements GPR, specifically, a procedure capable of making predictions about the value of an unknown function or model g (at a new input location $z_{\tau+1}$) given a set of observations $\{\boldsymbol{y}_\tau, \mathcal{Z}_\tau\}$. This procedure is summarized in Algorithm 1.

Hyperparameters of Gaussian Process Regression

Note that the kernel function used in Algorithm 1 may depend on many hyperparameters—if k is an RBF kernel, for instance (see the section "Gaussian Processes Priors"), then appropriate values for the set of hyperparameters $\mathcal{H}_p = \{A, \sigma_i, \omega_n\}$ must be selected. Although these values can be selected by a domain expert with prior knowledge of the functions being estimated, many techniques exist that automatically select values for GP hyperparameters. One

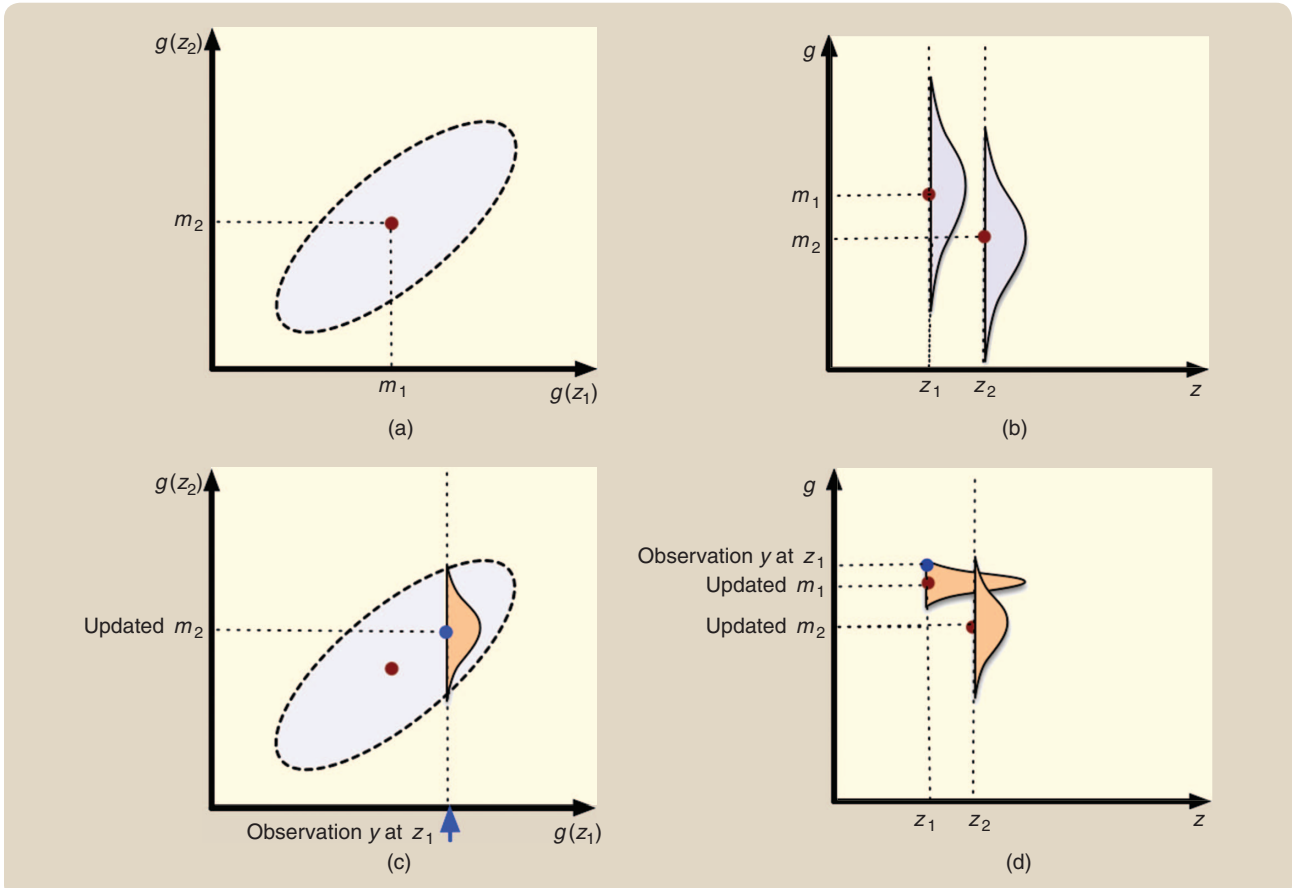

FIGURE 3 A pictorial illustration of posterior inference for Gaussian processes. (a) The joint prior distribution of outputs $g(\mathbf{z}_1)$ and $g(\mathbf{z}_2)$, associated with inputs $\mathbf{z}_1$ and $\mathbf{z}_2$, (b) the marginal prior distribution of output at locations $\mathbf{z}_1$ and $\mathbf{z}_2$, (c) the predictive distribution of output $g(\mathbf{z}_2)$ at location $\mathbf{z}_2$, conditioned on the observation of output y for input $\mathbf{z}_1$, and (d) the marginal posterior distribution of outputs at locations $\mathbf{z}_1$ and $\mathbf{z}_2$.

ALGORITHM 1 Gaussian process regression (GPR) algorithm.

Input: $\mathcal{Z}_\tau$ (inputs), $\boldsymbol{y}_\tau$ (targets), k (covariance function), w_n^2 (noise variance), $z_{\tau+1}$ (test input)
Output: $m(\mathbf{z}_{\tau+1})$ (predictive mean), $\Sigma(\mathbf{z}_{\tau+1})$ (predictive variance)
1: Compute $\mathbf{k}(\mathcal{Z}_\tau, \mathbf{z}_{\tau+1}) := [k(\mathbf{z}_1, \mathbf{z}_\tau), \ldots, k(\mathbf{z}_\tau, \mathbf{z}_{\tau+1})]$
2: Compute $\mu(\mathcal{Z}_\tau) = \frac{1}{\tau}\sum_{t=1}^{\tau} y(\mathbf{z}_t)$
3: Compute kernel matrix

$$\boldsymbol{K}(\mathcal{Z}_\tau, \mathcal{Z}_\tau) := \begin{bmatrix} k(\mathbf{z}_1,\mathbf{z}_1) & k(\mathbf{z}_1,\mathbf{z}_2) & \cdots & k(\mathbf{z}_1,\mathbf{z}_\tau) \\ k(\mathbf{z}_2,\mathbf{z}_1) & k(\mathbf{z}_2,\mathbf{z}_2) & \cdots & k(\mathbf{z}_2,\mathbf{z}_\tau) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{z}_\tau,\mathbf{z}_1) & k(\mathbf{z}_\tau,\mathbf{z}_2) & \cdots & k(\mathbf{z}_\tau,\mathbf{z}_\tau) \end{bmatrix}$$

4: $\boldsymbol{L} := \text{Cholesky}(\boldsymbol{K}(\mathcal{Z}_\tau, \mathcal{Z}_\tau) + \omega_n^2 \boldsymbol{I}_\tau)$ //Perform Cholesky decomposition
5: $\boldsymbol{\alpha}_\tau = \boldsymbol{L}^T \backslash (\boldsymbol{L} \backslash (\boldsymbol{y}_\tau - \mu(\mathbf{z}_\tau)\mathbf{1}_\tau))$ //GPR coefficients, (11)
6: $m(\mathbf{z}_{\tau+1}) = \mu(\mathbf{z}_{\tau+1}) + \boldsymbol{\alpha}_\tau^T \mathbf{k}(\mathcal{Z}_\tau, \mathbf{z}_{\tau+1})$ //Predictive mean, (10)
7: $\boldsymbol{v} := \boldsymbol{L} \backslash \mathbf{k}(\mathcal{Z}_\tau, \mathbf{z}_{\tau+1})$
8: $\Sigma(\mathbf{z}_{\tau+1}) := k(\mathbf{z}_{\tau+1}, \mathbf{z}_{\tau+1}) + \omega_n^2 - \boldsymbol{v}^T \boldsymbol{v}$ //Predictive variance, (12)

such technique is based on maximum likelihood estimation: a procedure that selects hyperparameter values by maximizing the likelihood of the observed outputs given hyperparameters $p(\boldsymbol{y}_\tau | \boldsymbol{g}_\tau, \mathcal{H}_p)$. This procedure requires that the point estimates of g are constructed, which may be difficult to obtain. An alternative technique maximizes the marginal likelihood $p(\boldsymbol{y}_\tau | \mathcal{H}_p)$, where the latent Gaussian vector $\boldsymbol{g}_\tau$ is marginalized out. A benefit of the latter procedure is that it moves the inference process one level up the Bayesian hierarchy. Since the probability distributions over the possible values of $\boldsymbol{g}_\tau$ (instead of the point estimates) are considered, the chance of overfitting is reduced. Finally, another technique for hyperparameter estimation is based on Bayesian inference methods, such as Markov chain Monte Carlo approaches. These Bayesian methods have an additional advantage of providing uncertainty estimates over the estimated hyperparameters. However, they can be computationally expensive.

Sparse Gaussian Processes for Online Settings

The previous section assumed that the entire set of τ noisy observations of g can be stored in memory. In practice, this set can become unbounded as a system evolves and more

measurements are taken, which is troublesome, since using GPs to predict the mean of the underlying function g [according to (10)] requires an expensive matrix inversion operation [which scales as $O(\tau^3)$ with the number τ of observed samples]. This is referred as the *batch prediction setting*. Many sparsification schemes have been developed to reduce the computational complexity of batch prediction using GPs to $O(\tau m^2)$, where m is the number of reduced parameters. One possible GP sparsification approach consists of a sequential method that incrementally builds a dictionary of basis vectors that serves as a representative summary of the full training set [52]. Such a dictionary is used to adequately describe the input domain without including a basis point at the location of every observed data point. This setting, therefore, is useful when collecting new measurements and determining whether to add them to a training set.

A related sparsification challenge in utilizing GPs occurs in the *online setting*: Even if one carefully selects which points to add to the training set, it is still possible that the set grows too large and cannot be kept in memory. In this case, a criterion must be defined to determine which points may be removed from the training set (that is, those whose size $|\mathcal{Z}|$ exceeds a given "budget"), while retaining information about the measurements collected up to that point. Addressing this latter problem requires identifying ways in which to approximate a given GP with a smaller set of bases.

Most existing sparsification approaches that address the above-mentioned issues are based on the observation that the training set $\mathcal{Z}$ generates a family of functions $\mathcal{F}_{\mathcal{Z}} \subset \mathcal{H}$ (where $\mathcal{H}$ is the space of functions of interest) whose richness characterizes the quality of the posterior inference. A natural and simple method to determine whether to add a new point to the subspace $\mathcal{F}_{\mathcal{Z}_\tau}$ spanned by $\{\psi(z_1), \ldots, \psi(z_\tau)\}$ is to check how well it is approximated by the elements in $\mathcal{Z}$. This is known as the kernel linear independence test [53], which is realized by computing

$$\beta_{\tau+1} = \min_{\alpha_\tau} \left\| \sum_{i=1}^{\tau} \boldsymbol{\alpha}_\tau[i]\,\psi(z_i) - \psi(z_{\tau+1}) \right\|_{\mathcal{H}}^2, \tag{14}$$

where $\beta_{\tau+1}$ is a scalar denoting the length of the residual of $\psi(z_{\tau+1})$ projected onto the subspace $\mathcal{F}_{\mathcal{Z}_\tau}$, the subspace of functions that can be represented by GPs (see Figure 4 as an illustration). The coefficient vector $\boldsymbol{\alpha}_\tau$ minimizing (14) is given by $\boldsymbol{\alpha}_\tau = \boldsymbol{K}_{\mathcal{Z}_\tau}^{-1} \boldsymbol{k}_{z_{\tau+1}}$,

$$\beta_{\tau+1} = k(z_{\tau+1}, z_{\tau+1}) - \mathbf{k}(\mathcal{Z}_\tau, z_{\tau+1})^T \boldsymbol{\alpha}_\tau. \tag{15}$$

When $\beta_{\tau+1}$ is larger than a specified threshold β_{tol}, a new data point is added to the dictionary. Otherwise, the weights $\boldsymbol{\alpha}_\tau$ (11) are updated, but the dimensionality of $\boldsymbol{\alpha}_\tau$ remains the same. When incorporating a new data point into the GP model, the inverse kernel matrix can be recomputed with a rank-1 update. The dictionary of data points selected by the above procedure is the *basis vector set*

$$\mathcal{BV} = \{z_1, \ldots, z_\tau\}. \tag{16}$$

When the budget of sample points in the training set is exceeded, a basis vector element in $\mathcal{BV}$ must be removed prior to adding another element [53]. Heuristics such as the oldest point (OP) method (which directly deletes the oldest vector by prioritizing the temporal locality of the data for the approximation) could be used. OP is fast; however, it does not guarantee that the most informative data points are preserved. The other (more structured) approach is to utilize information theoretic metrics or leverage the structure of the Hilbert space the model is embedded in to ensure sparsity. One such method, denoted **KL**, employs the *sparse online GP* algorithm [52]. This algorithm efficiently approximates the Kullback–Leibler (KL) divergence between the current GP and the $(\tau+1)$ alternative GPs missing one data point each and then deletes the data point with the largest KL divergence.

Here, we describe the sparse online GP algorithm [52] adapted for GP-MRAC. Let $p_{\max}$ denote the maximum desired size of $\mathcal{BV}$. At time $\tau+1$, given a new data point $z_{\tau+1}$, the budgeted sparse online GPR algorithm minimizes the KL divergence between the GPR model with the data point included, and the $\tau+1$ models with one data point deleted. To compute the updates in an online fashion, the algorithm first defines the following scalar quantities:

$$q^{(\tau+1)} = \frac{y - \boldsymbol{\alpha}_\tau^T \boldsymbol{k}_{x_\tau}}{\omega_n^2 + \boldsymbol{k}_{x_\tau}^T \boldsymbol{C}_\tau \boldsymbol{k}_{x_\tau} + k_\tau^*}, \tag{17}$$

$$b^{(\tau+1)} = -\frac{1}{\omega_n^2 + \boldsymbol{k}_{x_\tau}^T \boldsymbol{C}_\tau \boldsymbol{k}_{x_\tau} + k_t^*}, \tag{18}$$

where $\boldsymbol{\alpha}_\tau, \boldsymbol{k}_{x_\tau}$, and $\boldsymbol{C}_\tau = [\boldsymbol{K}_\tau + \omega_n^2 \boldsymbol{I}]^{-1}$ are defined in (10) and (12). Let $\boldsymbol{o}_{\tau+1}$ be the $(\tau+1)$ coordinate vector $(\boldsymbol{o}_{\tau+1}(\tau+1) = 1, \boldsymbol{o}_{\tau+1}(t) = 0,$ for all $t \neq \tau+1)$. $\boldsymbol{T}_{\tau+1}(\cdot)$ and $\boldsymbol{U}_{\tau+1}(\cdot)$ denote operators that extend a τ-dimensional vector and matrix to a $(\tau+1)$-dimensional vector and $(\tau+1)\times(\tau+1)$ matrix by appending zeros to them. The GP parameters can be solved recursively using

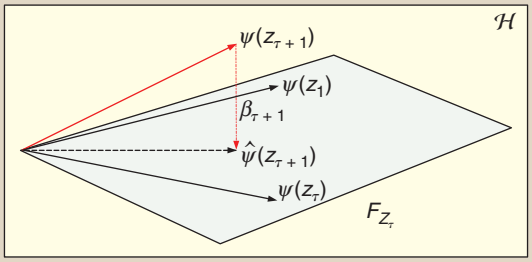

FIGURE 4 An example of the projection of $\psi(z_{\tau+1})$ onto the subspace F_{Z_τ} spanned by $\{\psi(z_1), \ldots, \psi(z_\tau)\}$. The scalar $\beta_{\tau+1}$ is the length of the residual and is a measure of the independence of $\psi(z_{\tau+1})$ with respect to F_{Z_τ}.

$$\boldsymbol{\alpha}_{\tau+1} = \boldsymbol{T}_{\tau+1}(\boldsymbol{\alpha}_\tau) + q^{(\tau+1)}\boldsymbol{s}_{\tau+1}, \tag{19}$$

$$\boldsymbol{C}_{\tau+1} = \boldsymbol{U}_{\tau+1}(\boldsymbol{C}_\tau) + b^{(\tau+1)}\boldsymbol{s}_{\tau+1}\boldsymbol{s}_{\tau+1}^T, \tag{20}$$

$$\boldsymbol{s}_{\tau+1} = \boldsymbol{T}_{\tau+1}(\boldsymbol{C}_\tau \boldsymbol{k}_{x_{\tau+1}}) + \boldsymbol{o}_{\tau+1}. \tag{21}$$

The inverse of the Gram matrix, denoted by $\boldsymbol{Q}$, needed to solve for $\beta_{\tau+1}$ defined in (15) is updated online using

$$\boldsymbol{Q}_{\tau+1} = \boldsymbol{U}_{\tau+1}(\boldsymbol{Q}_\tau) + \beta_{\tau+1}^{-1}(\boldsymbol{T}_{\tau+1}(\hat{e}_{\tau+1}) - \boldsymbol{o}_{\tau+1})(\boldsymbol{T}_{\tau+1}(\hat{e}_{\tau+1}) - \boldsymbol{o}_{\tau+1})^T, \tag{22}$$

where

$$\hat{e}_{\tau+1} \coloneqq \boldsymbol{Q}_\tau \boldsymbol{k}_{z_{\tau+1}}. \tag{23}$$

Finally, to delete a basis vector from the basis set, the model parameters are computed using the $(\tau+1)$th point, and the basis vector with the smallest score measure is chosen, where the score is given by

$$\epsilon_i = \frac{|\alpha_{\tau+1}[i]|}{\boldsymbol{Q}_{\tau+1}[i,i]}. \tag{24}$$

Let ι be the basis vector chosen to be discarded by the score (24). The deletion equations are

$$\hat{\boldsymbol{\alpha}} = \hat{\boldsymbol{\alpha}}^{\neg\iota} - \alpha^* \frac{\boldsymbol{Q}^*}{q^*}, \tag{25}$$

$$\hat{\boldsymbol{C}} = \boldsymbol{C}^{\neg\iota} + c^* \frac{\boldsymbol{Q}^*\boldsymbol{Q}^{*T}}{q^{*2}} - \frac{1}{q^*}[\boldsymbol{Q}^*\boldsymbol{C}^{*T} + \boldsymbol{C}^*\boldsymbol{Q}^{*T}], \tag{26}$$

$$\hat{\boldsymbol{Q}} = \boldsymbol{Q}^{\neg\iota} - \frac{\boldsymbol{Q}^*\boldsymbol{Q}^{*T}}{q^*}, \tag{27}$$

ALGORITHM 2 The budgeted sparse Gaussian process algorithm.

```
while new measurements (z_{τ+1}, y_{τ+1}) are available do
  Compute q^{(τ+1)}, b^{(τ+1)}, k*_{τ+1}, k_{z_{τ+1}}, ê_{τ+1} and β_{τ+1}.
  if β_{τ+1} < ε_tol then
    Compute ê_{τ+1} in (23) without extending the length of
    the parameters α and C.
    Perform a reduced update for α and C using (10)
    and (12).
  else
    Perform the update in (19) using the (τ+1)
    coordinate vector o_{τ+1}.
    Add the current input z_{τ+1} to the BV set.
    Compute the Gram matrix inverse Q_{τ+1} using (22).
    if |BV| > p_max then
      Compute scores {ε_i} for the candidate BV's using
      (24).
      Find ι = BV_d, where d = argmin_i ε_i.
      Delete ι and update α̂, Ĉ, and Q̂ using (25) to (27)
      respectively.
    end if
  end if
end while
```

where α^* is the ιth component in the vector $\boldsymbol{\alpha}_{\tau+1}$, and $\boldsymbol{\alpha}^{\neg\iota}$ represents the remaining vector. Similarly, $\boldsymbol{C}^{\neg\iota}$ ($\boldsymbol{Q}^{\neg\iota}$) represents the $\tau\times\tau$ submatrix in the $(\tau+1)\times(\tau+1)$ matrix $\boldsymbol{C}_{\tau+1}$ ($\boldsymbol{Q}_{\tau+1}$) associated with the basis vectors being kept, c^* (q^*) represents the (ι,ι) index in the matrix chosen by the score measure, and $\boldsymbol{C}^*(\boldsymbol{Q}^*)$ is the remaining τ-dimensional column vector. Using (10)–(27), the budgeted sparse GP algorithm is summarized by Algorithm 2.

The previous section presented the basic principle of GPR given noisy samples and illustrated how to predict the most likely values of an unknown function given the learned GP models. Based on these results, the next section presents sample applications of GPs being used to model and represent many functions of interest to the control community, such as system dynamic models and control policies.

Strengths of Gaussian Process Learning

At this point, it is instructive to compare the GP BNPM with more traditional parametric models, such as RBFNs or single hidden-layer NNs (SHL-NNs), and models with a large number of parameters, such as deep NNs (DNNs). The GP model is nonparametric, which means that the number of parameters in the model is not fixed a priori; rather, they grow with the data. This is in stark contrast with traditional RBFNs, where the number of RBFs and their centers must be specified a priori, or SHL-NNs, where the number of hidden-layer nodes must be specified a priori. The accuracy of RBFN representations is known to depend heavily on the a priori selection of the RBFN centers [24], [48], [49], [51]. Specifically, the representation is accurate only over areas of the state space where the RBFN centers are placed.

Conversely, GPs select their kernels from the data they are presented with. This ensures that the kernel dictionary is continuously updated as the system evolves in different areas of the state space. However, the GP weight-learning problem (with fixed kernel hyperparameters) is convex, and it can be addressed with regularized, recursive least squares when the noise is Gaussian. This is a distinct advantage over SHL-NNs and DNNs, where the weight-learning problem is nonconvex and stochastic gradient descent (backpropagation) is utilized [54], [55]. Furthermore, GPs address stochasticity naturally. GPs are models for distributions over functions, therefore, a GP can be treated as a *generative* model. That is, a draw from a GP is a smooth function. Each draw is different from the other, but all draws have the same functional mean and variance, as encoded by the kernels.

As such, GPs work exceedingly well for regression problems where the underlying function is smooth. The convex formulation of the GP weight-learning problem makes it highly amenable to learn the underlying function with relatively little data. However, DNNs (and in particular convolutional DNNs) are overparameterized, that is, they have

a very large number of parameters that must be tuned. As a result, large data sets are needed. Note that DNNs are extremely powerful techniques in their own right. However, how to use DNNs in online learning schemes where stability is of paramount importance is an open question. In contrast, GPs are more amenable to online and adaptive control applications, as described in this tutorial.

APPLICATION I: GAUSSIAN PROCESSES FOR MODEL REFERENCE ADAPTIVE CONTROL

This section describes the use of GPs in an MRAC setting. The conventional MRAC architecture is shown in Figure 5. In MRAC, the goal is to synthesize an adaptive controller that guarantees that a system with uncertain dynamics tracks a designer-provided reference model. This tutorial focuses on the approximate model inversion (AMI)-MRAC formulation, which is an approximate feedback-linearization-based MRAC method that allows the design of adaptive controllers for a general class of nonlinear plants (see for example, [56]–[60]). Let $\boldsymbol{x}(t) = [\boldsymbol{x}_1^T(t), \boldsymbol{x}_2^T(t)]^T \in \mathcal{D}_x \subset \mathbb{R}^n$ be the state of the system, such that $\boldsymbol{x}_1(t)$ and $\boldsymbol{x}_2(t)$ are elements of $\mathbb{R}^{n_x}$ and $n = 2n_x$. Let $\boldsymbol{u}(t) \in \mathcal{D}_u \subset \mathbb{R}^{n_x}$ be a control variable, and consider the following multiple-input, controllable, control-affine, nonlinear, uncertain, dynamical system

$$\begin{aligned} \dot{\boldsymbol{x}}_1(t) &= \boldsymbol{x}_2(t), \\ \dot{\boldsymbol{x}}_2(t) &= \boldsymbol{f}(\boldsymbol{x}(t)) + \boldsymbol{b}(\boldsymbol{x}(t))\boldsymbol{u}(t). \end{aligned} \tag{28}$$

Both $\boldsymbol{f}: \mathbb{R}^n \to \mathbb{R}^{n_x}$ and $\boldsymbol{b}: \mathbb{R}^n \to \mathbb{R}^{n_x \times n_x}$ are vector functions of the state $\boldsymbol{x}(t)$. We assume $\boldsymbol{f}(\boldsymbol{0}) = \boldsymbol{0}$. Both $\boldsymbol{f}$ and $\boldsymbol{b}$ are unknown functions assumed to be Lipschitz over a domain $\mathcal{D}$, and the control input $\boldsymbol{u}$ is assumed to be piecewise continuous so as to ensure the existence and uniqueness of the solution to (28) over $\mathcal{D}$.

The AMI-MRAC approach used is based on feedback linearization: a *pseudocontrol* input $\boldsymbol{\nu}(t) \in \mathbb{R}^{n_x}$ is designed to achieve a desired $\dot{\boldsymbol{x}}_2$. Given a desired pseudocontrol input $\boldsymbol{\nu}(t) \in \mathbb{R}^{n_x}$, a control command $\boldsymbol{u}$ can be found by approximate dynamic inversion. If the exact plant model $\boldsymbol{f}$ in (28) is known and invertible, the required control input to achieve the desired acceleration is computable by inverting the plant dynamics. However, since this usually is not the case, an approximate inversion model $\hat{\boldsymbol{f}}(\boldsymbol{x}) + \hat{\boldsymbol{b}}(\boldsymbol{x})\boldsymbol{u}$, where $\hat{\boldsymbol{b}}$ is a designer-chosen invertible function over all $\boldsymbol{x}(t) \in \mathcal{D}_x$, is employed in accordance with the AMI-MRAC framework [58], [59].

The control input $\boldsymbol{u}(t)$ can be found using approximate dynamic inversion

$$\boldsymbol{u} = \hat{\boldsymbol{b}}^{-1}(\boldsymbol{x})(\boldsymbol{\nu} - \hat{\boldsymbol{f}}(\boldsymbol{x})). \tag{29}$$

Let $\boldsymbol{z} = (\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \boldsymbol{u}^T)^T \in \mathbb{R}^{n+n_x}$ for brevity. The use of an approximate model introduces modeling error Δ in the system

$$\dot{\boldsymbol{x}}_2 = \boldsymbol{\nu}(\boldsymbol{z}) + \Delta(\boldsymbol{z}), \tag{30}$$

with

$$\Delta(\boldsymbol{z}) = \boldsymbol{f}(\boldsymbol{x}) - \hat{\boldsymbol{f}}(\boldsymbol{x}) + (\boldsymbol{b}(\boldsymbol{x}) - \hat{\boldsymbol{b}}(\boldsymbol{x}))\boldsymbol{u}. \tag{31}$$

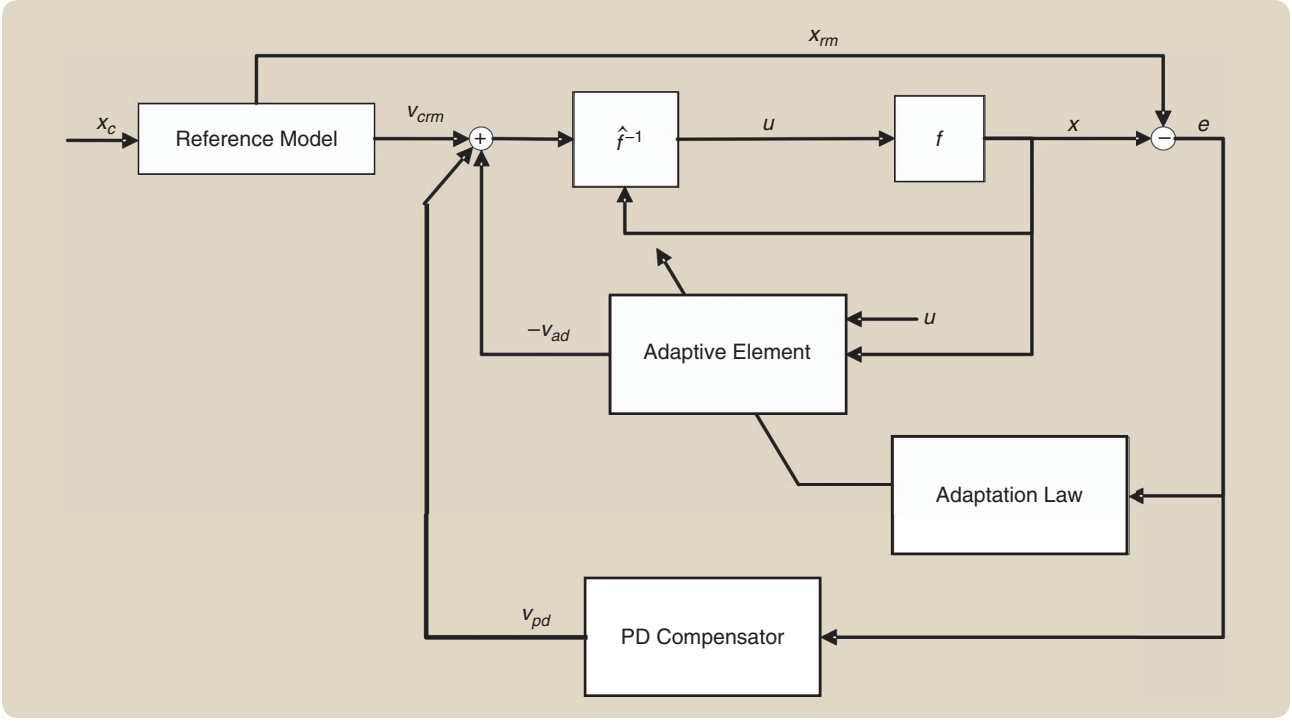

FIGURE 5 Conventional model reference adaptive control architecture. PD: proportional differential.

This tutorial presents a self-contained introduction to GPR techniques and discusses several examples of their use in solving control-related problems.

If $\boldsymbol{b}$ is both known and invertible with respect to $\boldsymbol{u}$, then an inversion model exists such that the modeling error is not dependent on the control input $\boldsymbol{u}$. In MRAC, a reference model is utilized by the designer to specify the desired response of the closed-loop system. The reference model is

$$\begin{aligned}\dot{\boldsymbol{x}}_{1_{rm}} &= \boldsymbol{x}_{2_{rm}},\\ \dot{\boldsymbol{x}}_{2_{rm}} &= \boldsymbol{f}_{rm}(\boldsymbol{x}_{rm}, \boldsymbol{r}),\end{aligned} \tag{32}$$

where $\boldsymbol{x}_{i_{rm}}$ and $\boldsymbol{f}_{rm}(\boldsymbol{x}_{rm}, \boldsymbol{r})$ denote the reference model states and dynamics, respectively, and $\boldsymbol{f}_{rm}(\boldsymbol{x}_{rm}, \boldsymbol{r})$ is assumed to be continuously differentiable with respect to $\boldsymbol{x}_{rm}$ for all $\boldsymbol{x}_{rm} \in \mathcal{D}_x \subset \mathbb{R}^n$. The command $\boldsymbol{r}(t)$ is assumed to be bounded and piecewise continuous. Furthermore, $\boldsymbol{f}_{rm}$ is assumed to be such that $\boldsymbol{x}_{rm}$ is bounded for a bounded input $\boldsymbol{r}(t)$.

The tracking error between the reference model and the state of the system is defined as $e(t) = \boldsymbol{x}_{rm}(t) - \boldsymbol{x}(t)$. The pseudocontrol input $\boldsymbol{\nu}$ is chosen to be

$$\boldsymbol{\nu} = \boldsymbol{\nu}_{rm} + \boldsymbol{\nu}_{\text{pd}} - \boldsymbol{\nu}_{\text{ad}}, \tag{33}$$

consisting of a linear feedforward term $\boldsymbol{\nu}_{rm} = \dot{\boldsymbol{x}}_{2_{rm}}$, a linear feedback term $\boldsymbol{\nu}_{\text{pd}} = [\boldsymbol{K}_1, \boldsymbol{K}_2]\boldsymbol{e}$ with $\boldsymbol{K}_1 \in \mathbb{R}^{n_x \times n_x}$ and $\boldsymbol{K}_2 \in \mathbb{R}^{n_x \times n_x}$, and an adaptive term $\boldsymbol{\nu}_{\text{ad}}(\boldsymbol{z})$ that is designed to cancel the modeling error Δ. For $\boldsymbol{\nu}_{\text{ad}}$ to cancel the modeling error Δ, assume there exists a unique fixed-point solution to $\boldsymbol{\nu}_{\text{ad}} = \Delta(\boldsymbol{x}, \boldsymbol{\nu}_{\text{ad}})$, for all $\boldsymbol{x} \in D_x$. This assumption implicitly requires the sign of control effectiveness to be known [59], that is, it requires the designer to know in which direction the control surface will move when control is applied, even though the exact relationship is unknown. For most systems, this is a reasonable assumption. Sufficient conditions for satisfying this assumption are discussed in [58] and [59].

Let

$$\boldsymbol{A} = \begin{bmatrix} 0 & \boldsymbol{I} \\ -\boldsymbol{K}_1 & -\boldsymbol{K}_2 \end{bmatrix}, \text{ and } \boldsymbol{B} = \begin{bmatrix} 0 \\ \boldsymbol{I} \end{bmatrix}.$$

From (33), the tracking error dynamics are

$$\dot{e} = \boldsymbol{A}e + \boldsymbol{B}[\boldsymbol{\nu}_{\text{ad}}(\boldsymbol{z}) - \Delta(\boldsymbol{z})]. \tag{34}$$

The baseline full-state feedback controller $\boldsymbol{\nu}_{\text{pd}}$ is chosen to make $\boldsymbol{A}$ Hurwitz. Hence, for any positive-definite matrix $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$, a positive-definite solution $\boldsymbol{P} \in \mathbb{R}^{n \times n}$ exists for the Lyapunov equation

$$\boldsymbol{0} = \boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} + \boldsymbol{Q}. \tag{35}$$

Note that the analysis henceforth holds for any control framework that can be brought to the form in (34), which includes traditional MRAC [13]–[15], [37].

Selection of Adaptive Element

We note that the system in (34) can be made exponentially stable if $\boldsymbol{\nu}_{\text{ad}}(\boldsymbol{z}) - \Delta(\boldsymbol{z}) = 0$ for all $\boldsymbol{z}$. From this viewpoint, a learning-based adaptive controller should try to minimize the difference between $\boldsymbol{\nu}_{\text{ad}}(\boldsymbol{z})$ and $\Delta(\boldsymbol{z})$ over all states. How this is accomplished depends on how much is known about the uncertainty. Two cases exist: 1) structured uncertainty and 2) unstructured uncertainty [61]. In the structured uncertainty case, it is assumed that $\Delta(\boldsymbol{z}) = \boldsymbol{G}^{*^T}\zeta(\boldsymbol{z})$. Note that $\boldsymbol{z} = (\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \boldsymbol{u}^T)^T$ where $\boldsymbol{G}^*$ is a k-dimensional vector of unknown weights, and ζ is a k-dimensional vector of known basis functions. This case is analyzed by traditional adaptive control [13], [14], [37], [62]. In particular, when learning-based methods are employed (such as those in [61], [63], and [64]), strong results for global exponential stability can be achieved [61]. The case of unstructured uncertainty arises when the form of the bases functions in ζ are unknown. This case is typically analyzed in the literature through NN based adaptive elements. In this tutorial, we focus on the case of unstructured uncertainty to provide an exact comparison between traditional NN-based methods and GP-based methods.

Consider the case of traditional MRAC, where Gaussian RBF networks [54] are used as adaptive elements [65]. The adaptive element $\boldsymbol{\nu}_{\text{ad}}$ of the control law (33) is represented by a linear combination of Gaussian RBFs: $\boldsymbol{\nu}_{\text{ad}}(\boldsymbol{z}) = \boldsymbol{W}^T\Phi(\boldsymbol{z})$, where $\boldsymbol{W} \in \mathbb{R}^{n_x \times L}$ and $\Phi(\boldsymbol{z}) = [1, \phi_2(\boldsymbol{z}), \phi_3(\boldsymbol{z}), \ldots, \phi_L(\boldsymbol{z})]^T$ is an L-dimensional vector of RBFs. Gaussian RBFs are universal approximators: They can model a continuous function over a compact domain to arbitrary accuracy, given L is sufficiently large [66]. For Gaussian RBFN, it can be shown that the weight update law is

$$\dot{\boldsymbol{W}} = \text{Proj}\left(-\boldsymbol{\Gamma}_W e^T\boldsymbol{P}\boldsymbol{B}\Phi(\boldsymbol{z})\right), \tag{36}$$

where $\boldsymbol{\Gamma}_W$ denotes a positive-definite-learning rate matrix, and the projection operator Proj (used to bound the weights

[67]) guarantees uniform ultimate boundedness of the tracking error and adaptive parameters if the system operates within the domain over which the RBF centers are distributed [13], [65]. This adaptive law does not guarantee that the ideal parameter matrix W^* convergences to the optimal parameter vector, unless a condition of *persistency of excitation* on the regressor function $\Phi(z)$ is satisfied [13], [32], [38].

Adaptive Control with Gaussian Process Adaptive Elements

In this section, GP-based MRAC is introduced, where GPs are used to learn and cancel the modeling error $\Delta(z)$ in (31). In traditional MRAC, it is assumed that Δ is a (smooth) deterministic function for which an input–output map in the form of an NN or RBFN is learned. The limitations of RBFN-based approaches have been discussed in the previous section. As an alternative to the deterministic fixed parameter models employed in RBFN, GP-MRAC [36] is introduced. In GP-MRAC, the modeling error is described by a time-varying (prior) mean and covariance function, and GPs are used to learn the modeling error as a continuous function of the time and state. As the system evolves and measurements are taken, Bayesian posterior updates are used to build a generative model of the modeling error. This new approach of learning utilizing probabilistic generative models offers significant benefits. In contrast to learning an RBFN with an a priori fixed set of bases, this approach can grow the set of bases during operation. Furthermore, this approach can inherently handle sensor noise, stochastic processes, and measurements effects such as servo chattering, external disturbances, and other nondeterministic effects. Furthermore, the tight integration with the MRAC framework offers significant benefits. The MRAC direct adaptive control framework enables the derivation of stability guarantees using stochastic control theory for switched uncertain systems, and it leads to an online learning framework that guarantees system stability during the learning stage. This should be contrasted with existing work in GP-based adaptive control (for example, [10], [41], [42], [44], and [45]), which has mostly been in the indirect adaptive control setting without any guarantees on stability.

For the sake of clarity of exposition, assume that $\Delta(z) \in \mathbb{R}$. One way to extend the approach to the multidimensional case is to utilize a vector-valued GP prior [68], [69]. When Δ follows a GP model, then

$$\Delta(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)), \tag{37}$$

where $m(\cdot)$ is the mean function, and $k(\cdot, \cdot)$ is a real-valued, positive-definite covariance kernel function. Given the model (37), in GP-MRAC, the modeling error is learned using a GP-based adaptive element ν_{ad}:

$$\hat{\nu}_{ad}(z) \sim \mathcal{GP}(\hat{m}(z), k(z, z')), \tag{38}$$

where $\hat{m}(z)$ is the estimate of the mean of (37) updated using (10).

The next section demonstrates the performance of GP-MRAC and compares it with other parametric methods through flight control experiments. For theoretical properties of GP-MRAC, see "Gaussian Process Model Reference Adaptive Control (GP-MRAC) Theoretical Guarantees."

Flight-Test Results with Gaussian Process Model Reference Adaptive Control on a Quadrotor

Flight-test results of the GP-MRAC architecture on a quadrotor unmanned aircraft system are presented here. These results first appeared in [72] and are reproduced here for completeness. In addition, several simulation results on GP-MRAC have been previously presented in the literature, including [36], [73], and [74]. An exhaustive comparison of GP-MRAC and RBFN-based MRAC with robustifying modification (projection operator) can be found in [36].

Three separate flight-test experiments with multiple trials were conducted to ensure statistically significant findings. The reference model tracking performance of GP-MRAC is compared with two variants of RBFN-MRAC as well as GP-MRAC with hyperparameter optimization [75]. In all cases, GP-MRAC achieves a significant performance advantage over traditional fixed-parameter RBFN-MRAC and concurrent-learning-based MRAC (CL-MRAC) [60]. The experiments were performed in the Aerospace Control Laboratory's RAVEN test environment at the Massachusetts Institute of Technology [76]. The RAVEN test environment uses a motion capture system to obtain accurate estimates of the position and attitude of autonomous vehicles. Accelerations and angular rates were calculated using a fixed-interval smoother. Position measurements are numerically differentiated twice to obtain accelerations, which leads to increased noise. The motion capture system does provide high-fidelity position measurements, which balances the noise introduced through numeric differentiation. However, it was observed that the GP was fairly robust to any noise introduced through numeric differentiation.

The flight experiments in this article were performed on the smaller of the two quadrotors shown in Figure 6. The vehicle weighs 96 g without the battery and measures 18.8 cm from motor to motor. Onboard attitude control is performed on a custom autopilot, with attitude commands being calculated at 1 kHz. Due to limitations of the speed controllers, the smaller quadrotor motors only accept motor updates at 490 Hz. More details on the autopilot and attitude control can be found in [78] and [79].

Augmentation of Baseline Linear Control with Gaussian Process-Based Adaptation

The outer loop position and velocity control of the baseline proportional-integral-derivative (PID) controller on the quadrotor was augmented with AMI-MRAC adaptive control. The outer loop of the baseline controller generates

Gaussian Process Model Reference Adaptive Control (GP-MRAC) Theoretical Guarantees

In most of the work on neuroadaptive control methods [13], [57], [65], [S1], [S3], the size of the ultimate bound on the tracking error has not been tightly quantified, mainly because the universal approximation theorems for radial basic function networks or neural network adaptive elements only guarantee the existence of a bounded error. This sidebar summarizes two key results that utilize the nonparametric nature of Gaussian processes (GPs) to quantify the ultimate bounds in terms of the kernel bandwidth, kernel dictionary size, and magnitude of outliers. These results first appeared in [36].

The first result presented here consists of an upper bound on the error between the best estimate of the GP mean (given the kernel budget) that is the maximum number of kernels allowed in the basis set and the ideal GP mean (given the available training data). Let the error between these two quantities be $\epsilon_m^\lambda(z) = m^\lambda(z) - \hat{m}^\lambda(z)$, and let $\tau^{-2}K_{ij} \coloneqq \tau^{-2}k(\mathbf{z}_i, \mathbf{z}_j)$ be at the (i,j)th element of a normalized kernel matrix associated with m^λ. The approximate mean is associated with a kernel matrix induced by a quantization operator $\vartheta : \{1, \ldots, \tau\} \to \{1, \ldots, p_{\max}\}$, such that $\mathbf{z}_i \mapsto \mathbf{c}_{\vartheta i}, \mathbf{c}_{\vartheta i} \in \mathcal{D}_x$ are the set of chosen bases $\mathcal{BV}$ Let $\hat{m}^\lambda(\mathbf{z}) = \Sigma_{i=1}^{\tau} \tilde{\alpha}_i k(\mathbf{c}_{\vartheta(i)}, \mathbf{z})$, where $\tilde{\boldsymbol{\alpha}} = (\tilde{\boldsymbol{K}} + \omega^2 \boldsymbol{I})^{-1}\boldsymbol{y}$ and where $\tilde{\boldsymbol{K}}_{ij} \coloneqq k(\mathbf{c}_{\vartheta i}, \mathbf{c}_{\vartheta j})$.

Theorem S1 (Global Approximation Theorem [36])
Let m^λ and $\hat{m}^\lambda p$ be defined as above and let $\|y\|_\infty \leq M^\lambda \in \mathbb{R}$. Then,

$$\left\| \epsilon_m^\lambda(\mathbf{z}) \right\| \leq \frac{2\kappa^2 M^\lambda \sqrt{k_{\max}}}{\omega^4} + \frac{\kappa k_{\max} M^\lambda}{\omega^2}, \tag{S4}$$

where $k_{\max} \coloneqq \max_i \| \psi(\mathbf{z}_i) - \psi(\mathbf{c}_{\vartheta(i)}) \|_{\mathcal{H}}$ is the largest kernel approximation error, κ is the maximum value of the chosen kernel (one for Gaussian kernel), and ω^2 is the variance of the measurement noise.

See [36] for a proof of Theorem S1. The key point to note about the above universal approximation result is that it relates the bound on $\epsilon_m^\lambda(\mathbf{z})$ directly to the cyberphysical quantities of interest: available computational power, available memory, and the quality of the sensor data. Specifically, the bound can be made tighter by reducing $k_{\max}$, which requires increasing the kernel budget to increase the density of the kernels and, therefore, requires more memory and computational power. The bound can also be reduced by decreasing the magnitude of the outliers M^λ, which may require selection of better sensors or filtering mechanisms.

A key point to note here is that the above result assumes Gaussian measurement noise. Therefore, even though increasing ω^2 may provide a weak mechanism to offset the effect of the outliers, it would also affect the prediction accuracy of the GP generative model. Note that GPs can also accommodate non-Gaussian noise [S4]–[S6]; in particular, the technique in [S7] has been shown to be a computationally effective means of handling non-Gaussian noise.

The second result described here provides a probabilistic ultimate bound on the tracking error that is directly proportional to ϵ_{tol}, which is the linear independence check tolerance used in the sparse online GP algorithm described in Algorithm 2.

desired acceleration to track a spline-based trajectory. These are then mapped to thrust and desired attitude commands [79]. The inner-loop attitude controller is implemented on board the vehicle with a PID controller and was left unaltered. The outer loop runs at 100 Hz on an offboard computer, and the inner loop attitude control runs at 500 Hz. Here, the attitude control is assumed to be sufficiently fast compared to the outer-loop controller.

The experiments are performed in a rapid control transfer setting [80], in which the well-tuned, PID, inner-loop and outer-loop controller from the larger quadrotor are implemented directly on the smaller quadrotor. The inner-loop controller is unaltered, while the outer-loop controller is augmented in the framework of AMI-MRAC using GP-MRAC and other methods. The large differences in size and mass between the two quadrotors result in relatively poor position and velocity tracking with the small quadrotor, when it uses the gains from the larger quadrotor. In addition, the lack of knowledge of the mapping of the speed controller commands to propeller thrust for the small quadrotor requires significant input from the altitude-integral PID term and leads to poor vertical and horizontal position tracking. In this AMI-MRAC implementation, the inversion law is only approximate, with modeling errors on the order of 2–6 (m/s^2) during experiments. Additionally, the PID controllers are tuned only to achieve stability and moderate performance. Hence, any gains in performance over the baseline controller are due to the adaptation.

In the following experiments, a baseline model and controller are used that are qualitatively similar to the true model, albeit with scaling differences. Note that since a GP is a nonparametric function approximator, the form of the model error does not matter. In what follows, baseline PID refers to the outer-loop PID controller on the smaller quadrotors with gains directly transferred from the larger (well-tuned) quadrotor. MRAC refers to the controller obtained by augmenting the baseline law with an RBFN adaptive element and CL-MRAC, a learning-based version of MRAC with RBFNs and gradient-based updates that concurrently use recorded and current data for adaptation and learning [60], [80], [81]. See [80] for details on how the adaptive element ν_{ad} is calculated using MRAC and CL-MRAC. Finally, the control law in (33) is slightly modified

Theorem S2 (Mean Square Uniform Boundedness of GP-MRAC [36])

Consider the system described by (28), the control law given by (29) and (33), and assume that the uncertainty $\Delta(z)$ is representable by a GP such as (37). Algorithm 3 and the adaptive signal $\nu_{\text{ad}}(z) = \hat{m}^{\lambda}(z)$ guarantee that the system is (almost surely) mean square, uniformly, and ultimately bounded in the set

$$\Theta_{\gamma}^{\lambda} = \left\{ \|e\| \geq \frac{c_5^{\lambda} M^{\lambda} + \sqrt{(c_5^{\lambda} M^{\lambda})^2 + 2\lambda_{\min}(\boldsymbol{Q}) c_1}}{\lambda_{\min}(\boldsymbol{Q})} \right\}, \tag{S5}$$

where

$$c_5^{\lambda} \coloneqq c_3 \sqrt{k_{\max}^{\lambda}} + c_4 k_{\max}^{\lambda}, \tag{S6}$$

and the size of the bound on the tracking error $\|e\|$ is proportional to ϵ_{tol}, $\lambda_{\min}(\boldsymbol{Q})$ is the minimum eigenvalue of the matrix $\boldsymbol{Q}$ satisfying the Lyapunov equation $\boldsymbol{0} = \boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} + \boldsymbol{Q}$, and $k_{\max} \coloneqq \max_i \|\psi(\boldsymbol{z}_i) - \psi(\boldsymbol{c}_{\vartheta(i)})\|_{\mathcal{H}}$ is the largest kernel approximation error.

See [36] for a proof of Theorem S2. The key point to note here is that the above bound provides an insight into how to perform a tradeoff between the computational budget available on board an embedded computer and the effort that must be spent in designing the baseline controller utilized in the approximate model inversion-MRAC framework to increase $\lambda_{\min}(\boldsymbol{Q})$. In particular, the tracking error bound gets tighter as the kernel density increases (with more memory and processing power), whereas a baseline controller [which results in a higher $\lambda_{\min}(\boldsymbol{Q})$] can be used to offset poor onboard computational power. That is, if a significant amount of computational power is available, then the baseline proportional-integral-derivative control need not be very "good" [with a larger $\lambda_{\min}(\boldsymbol{Q})$], whereas if the available computational power is restricted (thereby restricting learning and representational capabilities by limiting the maximum size of $\mathcal{BV}$), a better baseline controller may help reduce the tracking error bound.

Another key formal result relating the linear independence of $\mathcal{BV}$ and the excitation in the system was proven in [33], where it was shown that the linear independence of $\mathcal{BV}$ ensures that persistency of excitation (PE) in the state space is visible in $\mathcal{H}$ (that is, that the regressors are also PE). Since Algorithm 2 aims to enforce this independence subject to the tolerance ϵ_{tol}, PE is never lost [ensuring $\boldsymbol{K}(\boldsymbol{Z}_{\tau}, \boldsymbol{Z}_{\tau})$ is invertible].

REFERENCES

[S4] A. B. Chan and D. Dong. "Generalized Gaussian process models," in *Proc. Conf. Computer Vision and Pattern Recognition*, 2011, pp. 2681–2688.

[S5] P. Jylänki, J. Vanhatalo, and A. Vehtari, "Robust Gaussian process regression with a student-t likelihood," *J. Mach. Learn. Res.*, vol. 12, pp. 3227–3257, Nov. 2011.

[S6] M. Opper and C. Archambeau, "The variational Gaussian approximation revisited," *Neural Comput.*, vol. 21, no. 3, pp. 786–792, 2009.

[S7] D. Seiferth, G. Chowdhary, M. Muhlegg, and F. Holzapfel, "Online Gaussian process regression with non-Gaussian likelihood," in *Proc. IEEE American Control Conf.*, Seattle, WA, 2017, pp. 3134–3140.

so that $\nu_{\text{ad}} = \rho_{\text{MAP}}\hat{m}(z)$, where ρ_{MAP} is a Bayesian scaling factor, as described in [72].

This controller is called GP-MRAC, and in this framework, the hyperparameters of the Gaussian kernel (such as bandwidth and noise variance) are assumed static. It is possible to also adapt the hyperparameters online, which was accomplished in the GP-MRAC-HP method [75]. There are many other techniques for the hyperparameter update for GPs [26], [46], [82], [83]. However, most are in a batch setting and not online implementable. There are also gradient-based techniques that could be utilized to adjust the hyperparameters online [84]–[86]. The technique in [75] is utilized here for comparison since it is the closest in form and real-time implementability to GP-MRAC. In MRAC and CL-MRAC, a projection operator [67] was used to ensure robustness, although the limit of the projection operation was not reached in experiments. The projection operator is not required for GP-MRAC and GP-MRAC-HP.

The controllers were separated into three different loops, each one controlling a corresponding x, y, z position. The input to the RBF for MRAC was given by $z_t = [\dot{x}, \dot{y}, \dot{z}, \vec{q}]$, where $\vec{q}$ is the quaternion representation of vehicle attitude. In the

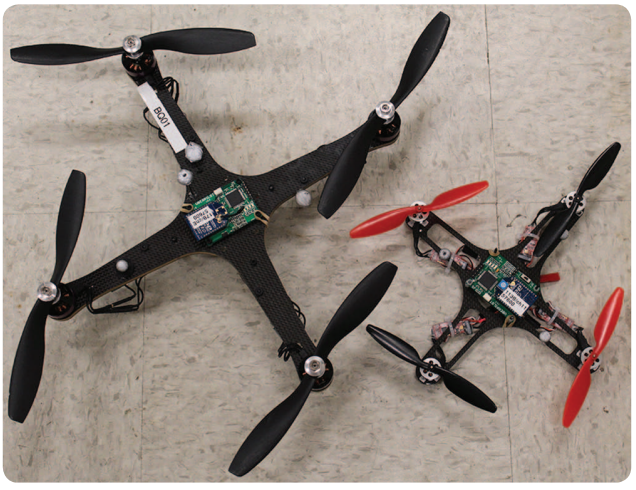

FIGURE 6 Two quadrotors equipped to fly in the Massachusetts Institute of Technology's Real-Time Indoor Autonomous Vehicle Test Environment (RAVEN) [77]. The baseline controller on both quadrotors is proportional integral derivative. The smaller quadrotor uses gains and thrust mappings from the larger one, resulting in relatively poor trajectory tracking performance. (Figure reproduced with permission from [72].)

results presented here, the input to the GP is $\boldsymbol{z}_t = [\dot{x}, \dot{y}, \dot{z}, \vec{q}, \nu]$. Adding the pseudocontrol input ν to the GP input was found to significantly reduce the tracking error in practice. However, adding ν to the RBFN-MRAC was found to degrade performance. One possible reason for this is that it is difficult to preallocate RBF centers over the commanded signals ν, since it is not entirely known how ν will behave. Additionally, the number of RBFs needed to uniformly cover the input domain increases exponentially with the number of dimensions of the input. While GP-MRAC is able to dynamically allocate centers over only active regions of the input domain, MRAC with fixed centers cannot adequately cover the domzain without an exponential number of centers.

Several different values of learning rates for RBFN-MRAC and CL-MRAC were analyzed through simulation and preliminary flight testing for MRAC. Note that GP-MRAC and GP-MRAC-HP do not require a learning rate. The results presented here correspond to values that resulted in good tracking performance without oscillations. The best initial MRAC learning rate was found to be $\Gamma_W = 2$, and the learning rate of the adaptive law that trains on recorded data was found to be $\Gamma_{W_b} = 0.5$. In theory, MRAC and CL-MRAC learning rates can remain constant in adaptive control. However, stochastic stability results indicate that driving the learning rates to zero is required for guaranteeing convergence in the presence of noise [87]. The problem with this approach is that the effect of adaptation would be eventually removed. Therefore, in practice, the learning rate is set to decay to a small constant to avoid unstable or oscillatory behaviors. Learning rates for MRAC and CL-MRAC were decayed by dividing them by 1.5 and 2, respectively, every 20 s. These learning rates decayed until they reach the values of $\Gamma_W = 0.5$ and $\Gamma_{W_b} = 0.001$.

ALGORITHM 3 The Gaussian process model reference adaptive control (GP-MRAC) algorithm.

Input: tolerance for kernel independent test ϵ_{tol}, budget $p_{\max}$

while new measurements $\boldsymbol{x}(t)$ available **do**
  Given $\boldsymbol{z}_{\tau+1}$, compute $\beta_{\tau+1}$ using (15)
  Estimate $\hat{\dot{\boldsymbol{x}}}_{2_{\tau+1}}$ using Kalman filter-based fixed-point smoother [70], [71].
  Compute $y_{\tau+1} = \hat{\dot{\boldsymbol{x}}}_{2_{\tau+1}} - \nu_{\tau+1}$
  if $\beta_{\tau+1} > \epsilon_{\text{tol}}$ **then**
    if $|\mathcal{BV}(\lambda)| > p_{\max}$ **then**
      Delete element in $\mathcal{BV}(\lambda)$ based on methods in the section of Sparse GPs for online settings
    end if
    Add $\boldsymbol{z}_{\tau+1}$ to $\mathcal{BV}$ and increase the index λ
  end if
  Calculate $\hat{m}_{\tau+1}$ and $\hat{\Sigma}_{\tau+1}$.
  Set $\nu_{\text{ad}} = \hat{m}_{\tau+1}$.
  Calculate pseudo control ν using (33).
  Calculate control input $\boldsymbol{u}$ using (29) and execute it.
end while

For MRAC and CL-MRAC, 100 RBF centers were generated using a uniform random distribution over a domain where the states were expected to evolve. The centers for the position and velocity for the x and y-axis were spread across $[-2,2]$. For the vertical z-axis, the position and velocity were spread across $[-0.5,0.5]$ and $[-0.6,0.6]$, respectively. The centers corresponding to the attitude quaternions were placed within $[-1,1]$. The bandwidth for each RBF is set to $\sigma_i = 1$. For GP-MRAC, centers are assigned dynamically by Algorithm 3, and it is not required to assign them a priori. The budget for the online dictionary of active bases was set to 100 and $\epsilon_{\text{tol}} = 0.0001$. An RBF kernel was used with initial bandwidth $\sigma = 2.2$ and noise $\omega_n = 2$. These values were obtained by running GP-MRAC-HP for one trial and using the final estimated optimal hyperparameters. For GP-MRAC-HP, the adaptation parameter $b[0] = 0.02$ was used. In each of the experiments, the initial bandwidth μ was varied to show that the performance of GP-MRAC-HP was not greatly affected by the prior choice of μ. For the Bayesian-mixing parameter, $\epsilon = 0.002$ was used, corresponding to a rise time of 5 s. As the number of data points in a region increased, the variance of the GP decreased and the GP changed less with successive points in those regions. As opposed to the RBFN-based approaches, the learning rates did not need to be specified manually with GP-MRAC to avoid overfitting or oscillations.

Flight-Test Results

The quadrotor performed five trials consisting of 20 figure-8 maneuvers with a period of 6.28 s, requiring approximately 125 s per trial. In the case of GP-MRAC with hyperparameter estimation, the initial bandwidth was set to $\sigma = 0.55, 1.1, 2.2, 4.4, 8.8$ for trials 1, 2, 3, 4, 5, respectively. In Figure 7, a plot of the trajectory in space is presented. Using the initial PID controller, the quadrotor is unable to track the figure 8. CL-MRAC performs better, although it overshoots at points, resulting in sharp corners. Lastly, GP-MRAC performs the best in terms of tracking error and capability of qualitatively matching the figure-8 shape. GP-MRAC is also the most consistent of all controllers, producing little variation between iterations of tracking the desired trajectory.

Figure 8 shows the windowed tracking error of each algorithm as a function of time. The baseline PID controller does not perform well in terms of tracking error, mainly due to the use of a poor approximate model that negatively impacts the feedforward component of the control signal. RBFN-MRAC performs better over time, but the convergence rate is slow. CL-MRAC converges relatively quickly and with less tracking error than traditional MRAC. GP-MRAC substantially outperforms all three previously mentioned controllers in terms of tracking error. GP-MRAC reduces the steady-state error of the PID controller by approximately 85%, outperforms RBFN-MRAC by a factor

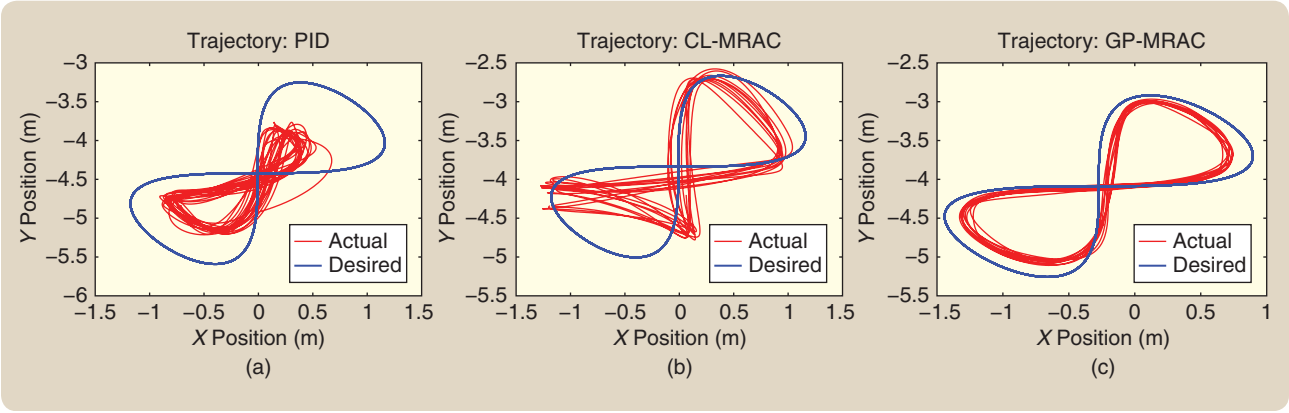

FIGURE 7 Sample trajectories of the quadrotor following a figure-8 pattern with (a) a baseline proportional-integral-derivative (PID) control, (b) concurrent learning model reference adaptive control (CL-MRAC), and (c) Gaussian process model reference adaptive control (GP-MRAC). The blue curve indicates the commanded path, and the red curve indicates the actual path flown by the quadrotor. GP-MRAC follows the trajectory best in terms of both tracking error and qualitatively matching the shape of the figure-8 trajectory. (Figure reproduced with permission from [72].)

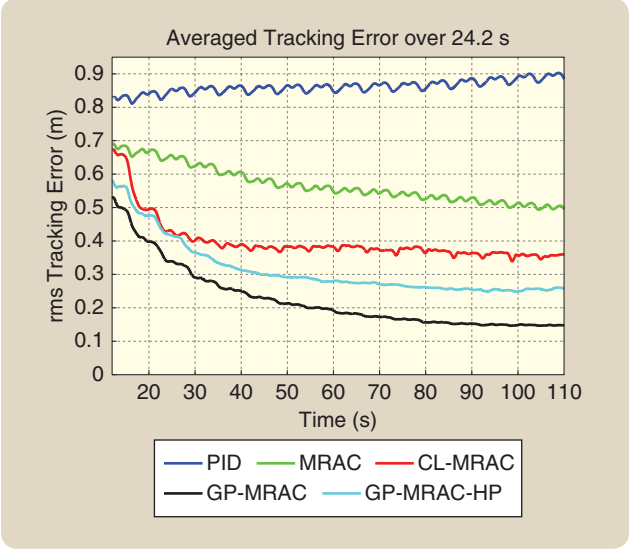

FIGURE 8 Gaussian process model reference adaptive control (GP-MRAC) outperforms traditional MRAC in terms of root mean square tracking error for a figure-8 trajectory. (Figure reproduced with permission from [72].)

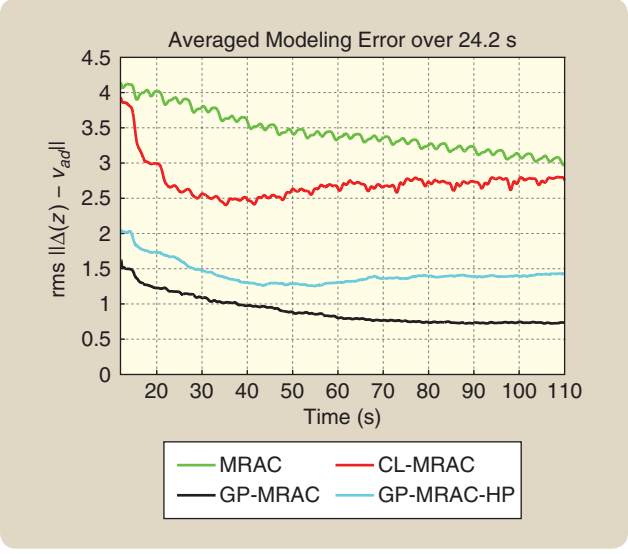

FIGURE 9 Gaussian process model reference adaptive control (GP-MRAC) outperforms traditional MRAC in terms of root mean square (rms) modeling error for a figure-8 trajectory. (Figure reproduced with permission from [72].)

of three, and outperforms CL-MRAC by a factor of two. GP-MRAC-HP with initial suboptimal bandwidth selection performs well, although not as well as GP-MRAC. As GP-MRAC-HP adapts the hyperparameters, the certainty in the adaptive controllers model of Δ decreases, leading to a lower ρ_{MAP} and higher tracking error. As time tends to infinity, GP-MRAC-HP converges to optimal hyperparameters, leading to the same performances as GP-MRAC with initial optimal hyperparameters.

Figure 9 presents the windowed root-mean-squared error of the modeling error $\|\Delta(z) - \nu_{\text{ad}}\|$. GP-MRAC models the error substantially better than RBFN-MRAC and CL-MRAC. GP-MRAC outperforms RBFN-MRAC by more than a factor of three. This improvement in characterizing the modeling error $\|\Delta(z) - \nu_{\text{ad}}\|$ is what fundamentally leads to the lower tracking error of GP-MRAC.

APPLICATION II: PLANNING AND OPTIMAL CONTROL—GAUSSIAN PROCESS DYNAMIC PROGRAMMING (GP-DP) AND MODEL PREDICTIVE CONTROL (MPC)

Besides adaptive control, GPs can also be used to solve multistage decision-making problems or optimal control and planning problems. One notable approach is GP dynamic programming (GP-DP) [44]. GP-DP is an approximate value-function-based DP algorithm, where the unknown value function is modeled by a GP, which allows for standard

DP techniques to be generalized to handle problems with continuous-valued states and actions. This section illustrates how GPs can be used with DP to solve planning and optimal control problems.

Gaussian Process Dynamic Programming

Assume a discrete-time optimal control problem with dynamics specified by $\boldsymbol{x}_{t+1} \sim p(\boldsymbol{x}_{t+1} | \boldsymbol{x}_t, \boldsymbol{u}_t)$ and per-step cost function by $c(\boldsymbol{x}_t, \boldsymbol{u}_t)$ (which measures the short-term cost or undesirability of executing action $\boldsymbol{u}_t$ when in state $\boldsymbol{x}_t$), where $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ are, respectively, the state and action at time t. This problem can be formulated as a Markov decision process (MDP) with model $\mathcal{M} = \{\mathcal{X}, \mathcal{U}, \mathcal{P}, c, \gamma\}$, where $\mathcal{X}$ and $\mathcal{U}$, respectively, are sets of states and actions; $\mathcal{P}$ and c, respectively, are dynamics and cost functions, and $\gamma \in [0, 1)$ is a discount factor, which determines the importance of future costs. An optimal state-value function V^* (which measures the expected total cost or long-term undesirability) is defined as

$$V_t^*(\boldsymbol{x}) = \min_{\boldsymbol{u}} \left\{ c(\boldsymbol{x}, \boldsymbol{u}) + \gamma \sum_{\boldsymbol{x}'} P(\boldsymbol{x}' | \boldsymbol{x}, \boldsymbol{u}) V_{t+1}^*(\boldsymbol{x}') \right\}, \tag{39}$$

for all states $\boldsymbol{x}$ and time steps $t = 1, \ldots, T-1$. The relation in (39) is known as the Bellman equation, optimality equation, or DP equation, which defines the value of a state $\boldsymbol{x}_t$ at a certain point in time in terms of the cost incurred by executing some action $\boldsymbol{u}$ when in that state, and the value of the possible subsequent states that result from executing that action. By calculating the value of all states, an optimal control policy $\pi^* : \boldsymbol{x} \to \boldsymbol{u}$ can be identified, which describes the optimal action as a function of the state. The computation of π^* is achieved through DP [88]. The state value of a policy π is defined by

$$V_t^{\pi}(\boldsymbol{x}) = \mathbb{E}\left[\sum_{\tau=0}^{\tau} \gamma^t c(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau = \pi(\boldsymbol{x}_\tau)) \middle| \boldsymbol{x}_0 = \boldsymbol{x} \right], \tag{40}$$

ALGORITHM 4 Classical dynamic programming, which assumes that the state transition probability p is known.

Input: p (state transition probability function), c (immediate cost function), $\mathcal{X}_{\text{DP}}$ (a set of states), $\mathcal{U}_{\text{DP}}$ (a set of controls), T (horizon)
Output: $\pi^*(\mathcal{X}_{\text{DP}}) := \pi_0^*(\mathcal{X}_{\text{DP}})$, optimal controls for $\mathcal{X}_{\text{DP}}$
1: $V_T^*(\mathbf{x}) = \min_{\boldsymbol{u}} c(\mathbf{x}, \boldsymbol{u})$, for all $\mathbf{x} \in \mathcal{X}_{\text{DP}}$, //initialize value function with terminal cost
2: **for** $t = T-1 : 0$ **do** //recursion
3: **for** $\mathbf{x}_i \in \mathcal{X}_{\text{DP}}$ **do**
4: **for** $\boldsymbol{u}_j \in \mathcal{U}_{\text{DP}}$ **do**
5: $Q_t^*(\mathbf{x}_i, \boldsymbol{u}_j) = c(\mathbf{x}_i, \boldsymbol{u}_j) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' | \mathbf{x}_i, \boldsymbol{u}_j) V_{t+1}^*(\mathbf{x}')$
6: **end for**
7: $\pi_t^*(\mathbf{x}_i) \in \operatorname{argmin}_{\boldsymbol{u} \in \mathcal{U}} Q_t^*(\mathbf{x}_i, \boldsymbol{u})$
8: $V_t^*(\mathbf{x}_i) = \min_{\boldsymbol{u}} Q_t(\mathbf{x}_i, \boldsymbol{u})$
9: **end for**
10: **end for**

where $V^*(\boldsymbol{x}) = \min_\pi V_t^\pi(\boldsymbol{x})$. Even though the state value function suffices to characterize optimality, it is useful to define the following action-state value function, also called the Q-function, as

$$Q_t^{\pi}(\boldsymbol{x}, \boldsymbol{u}) = c(\boldsymbol{x}, \boldsymbol{u}) + \gamma \sum_{\boldsymbol{x}'} P(\boldsymbol{x}' | \boldsymbol{x}, \boldsymbol{u}) V_{t+1}(\boldsymbol{x}'). \tag{41}$$

Note that as $T \to \infty$, V_t^π converges to the infinite horizon value function V^π and Q_t^π converges to the infinite horizon Q-function Q^π, which are defined as

$$V^{\pi}(\boldsymbol{x}) = \mathbb{E}_\pi\left[\sum_{\tau=0}^{\infty} \gamma^\tau c(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau) \middle| \boldsymbol{x}_0 = \boldsymbol{x} \right], \tag{42}$$

$$Q^{\pi}(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{E}_\pi\left[\sum_{\tau=0}^{\infty} \gamma^\tau c(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau) \middle| \boldsymbol{x}_0 = \boldsymbol{x}, \boldsymbol{u}_0 = \boldsymbol{u} \right]. \tag{43}$$

In the classic DP algorithm (Algorithm 4), the value function is defined over a finite set of actions (control values) $\mathcal{U}$ and finite set of states $\mathcal{X}$. The algorithm identifies an optimal plan (optimal actions for each possible state) via a DP approach. It first identifies the optimal action at the last step of a plan (time T). This is trivial, as the best action at this step is simply the one with minimum immediate cost c. Because state transitions terminate in the last step, the future value at the last step is zero ($V_{T+1} = 0$). As a consequence, the cost associated with a terminal state (the *terminal cost*) is simply the cost of the best action at that state. Line 1 of Algorithm 4 uses this observation to compute the value V of all states at time T. Identifying V_T is the base step of a DP approach. Specifically, it is possible to use V_T and apply backward recursion [and (39)] to compute the values of all states and actions one step earlier in the plan at $T-1$. This process can be repeated to all previous steps of a plan: $T-2, T-3, \ldots, 0$. By iterating over the steps of a plan, Algorithm 4 constructs a sequence of Q- and value functions for all possible states and actions at each possible decision stage; namely, Q_T and V_T, Q_{T-1} and $V_{T-1}, \ldots$, and Q_0 and V_0. At the end of this process, the algorithm will have computed Q_0 and V_0, which correspond to the Q- and value functions for all states at the first moment when the agent interacts with its environment. Note that the value function is computed for all states at each stage and that the optimization is performed in the action space; the total number of minimization steps needed, therefore, is $T|\mathcal{X}_{DP}||\mathcal{U}_{DP}|$. Although the classic DP algorithm is simple and instructive, it would be difficult to scale up when tackling real-world problems involving a possibly infinite number of states and control choices without forcing constraints or approximations.

To generalize the DP algorithm to problems with continuous state and control spaces, various function approximation techniques [17] may be used to approximately and compactly represent functions such as the value function or system dynamics. GP-DP uses GPs to estimate such functions based on samples or observations [44]. The computation details are summarized in Algorithm 5. GP-DP overcomes

one of the limitations of classical DP by modeling the value function V_k^* and the action-state value function Q_k^* as elements within the function space spanned by a GP prior. In particular, the state and action spaces are represented by the training inputs (supporting points in $\mathcal{U}$ and $\mathcal{X}$) for two value function GP models

$$V_k^*(\cdot) \sim \mathcal{GP}(m_v, k_v), \quad \text{and} \quad Q_k^*(\boldsymbol{x}, \cdot) \sim \mathcal{GP}(m_q, k_q). \tag{44}$$

Starting from the last control horizon, the estimated Q- and V-functions are iteratively constructed using the (given) transition and reward functions (corresponding to lines 9 and 13, respectively), much like in the standard DP algorithm. For the GP model in (44), the predictive distribution of $V_t^*(\boldsymbol{x}_*)$ for any state $\boldsymbol{x}_*$ can be obtained through (9). The expected future value $\mathbb{E}_V[V_{k+1}^*(g(\boldsymbol{x}, \boldsymbol{u}))]$ is obtained by the predictive mean of $V_k^*(g(\boldsymbol{x}, \boldsymbol{u}))$ given by (10). Moreover, the uncertainty over the value of states is naturally characterized by the GP predictive variance defined by (12). Because of the generalization of GPs, the requirement that state and action spaces are discrete is no longer necessary. Hence, GPs enable the V^*-function to generalize DP to continuous states, and enable the Q^*-function to generalize DP to continuous controls.

It is worth noting that, in the aforementioned DP algorithms, identifying optimal actions given a value function requires access to system dynamics and cost functions. However for many real-world problems, these functions may not be known a priori. In this case, RL is used to find the optimal policy. Specifically, due to model uncertainty, the system dynamics g is modeled probabilistically and must be learned through agent-environment interactions, so that the estimated model $\hat{g}$ can be incorporated into a planning algorithm such as GP-DP to solve an approximate optimal policy. This method is called model-based RL because an explicit estimation of dynamics is involved. Given the assumption that the dynamics evolve smoothly over time and are stationary, a GP model can be employed to estimate the short-term dynamics $g \sim \mathcal{GP}_g$. For each state dimension i, a separate GP model is trained, $(\boldsymbol{x}_{k+1}^i - \boldsymbol{x}_k^i) \sim GP(m_g, k_g)$, with the state-action pairs as training inputs and the difference between the successor state and the state in which the action was applied as an output. Given any test input $(\boldsymbol{x}_*, \boldsymbol{u}_*)$, the predictive distribution $p(g(\boldsymbol{x}_*, \boldsymbol{u}_*))$ is Gaussian distributed with mean and variance computed by (10) and (12), respectively. Note that the goal of GP-DP is to achieve approximate optimal control, which can be intractable when solving for long horizon problems. A computationally more feasible framework is GP-based model predictive controls (GP-MPC), where the state-transition dynamic model is updated online and the control command is computed based on a sliding time-step window, similar to receding-horizon GP-DP [2]. In an MPC setting, a GP model is computationally convenient and intuitive: when there is no training data (namely, when no observations of the system dynamics have been made) or the system is operating in nominal conditions, the system's behavior is captured by the prior g^0 over the expected change in state dimensions that follows from the execution of an action in a given state. If training data about the dynamics is subsequently collected, g can be adapted to fit the observed data. As an example, GP-MPC has been applied to fault-tolerant control problems [2] and outperformed parametric counterparts in their tested examples. Even though a detailed discussion of the general properties of MPC is beyond the scope of this work, it is worth emphasizing that MPC is a well-studied topic for which thorough tutorials exist and have been published in *IEEE Control Systems Magazine*, including [89]. The goal of this section, by contrast, was to demonstrate how GPs can be used in the MPC setting while relying on existing literature to provide the necessary background.

Finally GP-DP often requires an accurate value function approximation to derive an efficient policy, which may require many samples. For this reason, it might be challenging to apply GP-DP to high-dimensional problems. Although cases exist where inaccurate value function approximation is sufficient to determine efficient policies, this is often not the case. In [17], for instance, the authors note that approximate value functions will lead to suboptimal policies in approximate DP and RL. To address this curse-of-dimension issue, a sample-efficient, model-based RL framework called probabilistic inference for learning control (PILCO) [10] was developed to search for optimal policies directly in a policy space. This framework was evaluated on control problems ranging from cart-pole systems to robotic arms. Extensions of PILCO have been proposed that exploit problems where multifidelity simulators exist but can be sampled at different costs [90]. These methods have been applied to problems where a

ALGORITHM 5 Gaussian processes-dynamic programming, which assumes that deterministic dynamics g is known.

1: **Input:** $f, \mathcal{X}, \mathcal{U}$
2: **Output:** $\mathcal{GP}_v, \mathcal{X}, \pi^*(\mathcal{X}) \coloneqq \pi_0^*(\mathcal{X})$, optimal controls for $\mathcal{X}$
3: $V_T^*(\mathcal{X}) = \min_u c(\mathcal{X}, \boldsymbol{u})$ //Terminal cost
4: $V_T^*(\cdot) \sim \mathcal{GP}_v$ //GP model for V_T^*
5: **for** $t = T - 1$ to 0 **do**
6: **for** $\mathbf{x}_i \in \mathcal{X}$ **do**
7: **for** $\boldsymbol{u}_j \in \mathcal{U}$ **do**
8: $c(\mathbf{x}_i, \boldsymbol{u}_j) \sim \mathcal{N}(c, \sigma_g^2)$
9: $Q_t^*(\mathbf{x}_i, \boldsymbol{u}_j) = c(\mathbf{x}_i, \boldsymbol{u}_j) + \gamma \mathbb{E}_V[V_{t+1}^*(g(\mathbf{x}_i, \boldsymbol{u}_j))]$ //GP model for Q_t^*
10: **end for**
11: $Q_t^*(\mathbf{x}_i, \cdot) \sim \mathcal{GP}_q$
12: $\pi_t^*(\mathbf{x}_i) \in \operatorname{argmin}_u Q_t^*(\mathbf{x}_i, \boldsymbol{u})$
13: $V_t^*(\mathbf{x}_i) = Q_t^*(\mathbf{x}_i, \pi_t^*(\mathbf{x}_i))$
14: **end for**
15: $V_T^*(\cdot) \sim \mathcal{GP}_v$ //GP model for V_k^*
16: **end for**
17: **Return:** $\mathcal{GP}_v$.

limited number of real-world experiences are available and has been shown to successfully learn controllers for remote-controlled cars in challenging situations. This and many other applications of GPs to approximate optimal control and model-based RL problems exist, which suggests that GPs are flexible function approximators that can be used to solve real-world control problems where samples are expensive to collect and for which prior knowledge of the underlying dynamics may not exist. See [91] for a more comprehensive discussion about other applications involving model-based control and GPs.

APPLICATION III REINFORCEMENT LEARNING—GAUSSIAN PROCESS Q-LEARNING

The previous section discussed the application of GPs for solving planning problems where an optimal control policy must be identified and the system dynamics are known a priori (or can be estimated by sampling). RL determines an optimal policy $\pi^*(\boldsymbol{x}) = \text{argmax}_{\boldsymbol{u}} Q^*(\boldsymbol{x}, \boldsymbol{u})$ when dynamics $\mathcal{P}$ and/or *reward functions* r are unknown. Note that in the RL community, reward functions $r(\boldsymbol{x}, \boldsymbol{u})$ are often used in place of a cost function $c(\boldsymbol{x}, \boldsymbol{u})$; in this case, the min-operator in (39) becomes a max-operator. Reward functions measure the short-term desirability of executing action $\boldsymbol{u}$ when in state $\boldsymbol{x}$. Traditionally, RL methods have been categorized into two classes: model based and model free. As discussed in the previous section, model-based RL uses samples to learn the dynamics and reward functions associated with the system. Appropriate actions are then identified by a DP algorithm. Compared to model-free RL, model-based RL has been shown to provide better results with less samples [92] and is suitable for goal-directed actions [93]. Model-free RL algorithms, on the other hand, use samples to directly learn optimal control polices while not requiring the use of system models, and they may be more efficient when the policy space exhibits more regularity than the underlying dynamics. This section discusses the use of GPs to solve control problems within the *model-free*, value-based RL setting, effectively resulting in a method that nonparametrically extends the basis functions for Q-value functions.

Model-free, value-based RL methods can be used in both online and batch settings, and they can update an estimate of $Q(\boldsymbol{x}, \boldsymbol{u})$ based on these samples. In *online* RL, a decision maker selects and executes actions to sample trajectories from an environment or system whose dynamics are unknown. In *batch* RL, a collection of trajectories is provided to the learning algorithm. When an RL method learns the value function describing the value of states under the same policy used to collect samples, it is referred to as an *on-policy* algorithm; when the policy used to collect samples is different than the policy or behavior being learned, the algorithm is referred to as being *off policy*. The Q-learning algorithm [94] is one example of an off-policy technique; it executes an update of the form $Q_{t+1}(\boldsymbol{x}_t, \boldsymbol{u}_t) = Q_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \lambda_t \Delta_t$, where λ_t is a time-dependent learning rate and $\Delta_t = r(\boldsymbol{x}_t, \boldsymbol{u}_t) + \gamma \max_{\boldsymbol{u}'} Q_t(\boldsymbol{x}_{t+1}, \boldsymbol{u}') - Q_t(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the temporal difference (TD) error. One on-policy counterpart to Q-learning is the state-action-reward-state-action (SARSA) algorithm [95], which uses a different TD error-based update to the current policy: $\Delta_t = r(\boldsymbol{x}_t, \boldsymbol{u}_t) + \gamma Q_t(\boldsymbol{x}_{t+1}, \boldsymbol{u}_{t+1}) - Q_t(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Off-policy algorithms can be used in situations where batch data or robust, but suboptimal, policies are used for exploration (named safe-exploration policies). In particular, off-policy algorithms are capable of learning optimal control policies even when provided with samples or observations collected by an arbitrary policy and are therefore extremely valuable in situations where safe exploration policies are available and can be used to explore the system's dynamics. In the controls community, they have been recently explored for solving H_∞-optimal control problems [96] and multiple model control [97]. Off-policy Q-learning has been used for creating automated programs that play Atari games [98], albeit with DNNs instead of GPs, but the principle of training is the same as described below.

For continuous RL domains, linear-value-function approximation is often used to model the Q-function as a weighted combination of fixed bases $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u})$: $Q(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{\phi}^T(\boldsymbol{x}_t, \boldsymbol{u}_t)\boldsymbol{\theta}$. In this case, the off-policy gradient descent operation from Q-learning can be used to update $\boldsymbol{\theta}$ using the TD error $\Delta_t = r_t + \gamma \max_{\boldsymbol{u}'} \boldsymbol{\phi}^T(\boldsymbol{x}_{t+1}, \boldsymbol{u}')\boldsymbol{\theta}_t - \boldsymbol{\phi}_t^T\boldsymbol{\theta}_t$, where $\boldsymbol{\phi}_t = \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t)$. However, using function approximation in off-policy RL (including Q-learning and linear-least squares approaches [99], [100]) can cause divergence. It is possible to guarantee convergence of some off-policy algorithms under certain constraints, for instance, by ensuring that the initial policy is close to π^* [101]. The potential for divergence in off-policy settings, therefore, can limit the applicability of RL algorithms to safety-critical domains where divergence could be catastrophic.

This section introduces a model-free, off-policy, approximate RL technique, termed GPQ [102], [103], which uses a GP model to approximate action-state value functions and does not require a planner, such as GP-DP. Because GPQ is off policy, it can be used in both online or batch RL settings, independent of how the training data was obtained (that is, using a safe or completely exploratory or random policy). These properties hold even as new state features are added or less-important features are removed to maintain computational feasibility. Importantly, GPQ differs from other off-policy RL methods with fixed-parameter linear function approximation [104]–[107] since the basis functions in GPQ are automatically identified from data. Furthermore, it is possible to show that even though GPQ could, in pathological cases, diverge, formal analyses can be made that characterize the precise conditions in which this occurs [102]. Such analyses result in a method that converges to the best achievable optimal Q-function given that a regularization-like parameter is properly tuned (see "Demonstration of the Convergence Condition of Fitted Q-Learning").

Before describing the details of the GPQ method, it is worth noting that GPs have been previously used to solve

Demonstration of the Convergence Condition of Fitted Q-Learning

Typically, the parameter ω_n^2 is viewed as an uncorrelated, Gaussian measurement noise in Gaussian process (GP) literature. Alternatively, ω_n^2 can also be interpreted as a regularization term, which accounts for the fact that current measurements are not necessarily drawn from the true model and therefore prevents GPQ from converging too quickly to an incorrect estimate of Q^*. Therefore, ω_n^2 plays a pivotal role in preventing divergence. A counterexample is provided here to demonstrate divergence in the batch setting if ω_n^2 is insufficient and show convergence when ω_n^2 is large enough.

Consider a system with three nodes on the real line at locations –1, 0, and 1. At each time step, the agent can move deterministically to any node or remain at its current node. The reward associated with all actions is zero. All algorithms are initialized with $\hat{Q}(\mathbf{z}) = 1$ for all $\mathbf{z}, \gamma = 0.9999$, and an radial basis function kernel with bandwidth $\sigma = 1$ is used in all cases. Two settings of the regularization parameter, $\omega_n^2 = 0.1$ and $\omega_n^2 = 1$, are considered. Figure S1 shows that when ω_n^2 is set too low, the Bellman operator can produce divergence in the batch setting. If the regularization is set to the higher value, then the GP-fitted Q-iteration (GP-FQI) converges. It has been shown that determining the sufficient regularization parameter ω_n^2 depends only on the density of the data and the hyperparameters, not the initialization value of $\hat{Q}$ or γ [102].

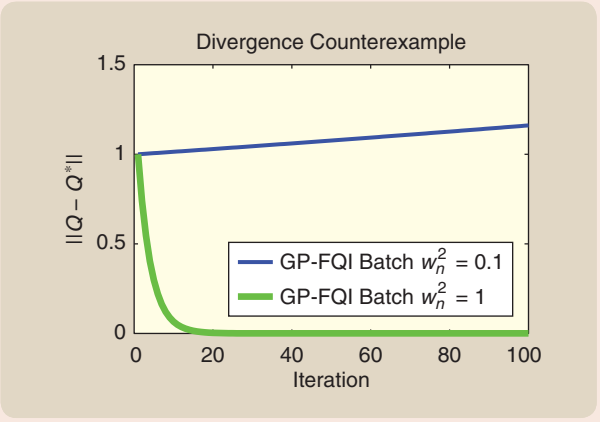

FIGURE S1 The maximum error $\|\hat{Q} - \hat{Q}^*\|$ is plotted for the Gaussian process-fitted Q-learning (GP-FQI) with insufficient regularization $\omega_n^2 = 0.1$ and sufficient regularization $\omega_n^2 = 1$. (Figure reproduced with permission from [102].)

RL problems. The GP-TD algorithm was originally proposed by Engel et al. [108]–[110]. The noteworthy difference between GPTD and GPQ is that GPTD imposes GP prior over state-value functions (V-functions) instead of state-action-value functions (Q-functions). We present GPQ in this tutorial because of its algorithmic simplicity. In addition to presenting the GPQ framework, sufficient conditions for convergence of GPQ to the optimal Q-function (Q^*) have been established, both for the batch and online settings (see "Convergence Analysis of GPQ"), which were lacking in the GP-TD methods proposed by Engel et al. [108]–[110].

The next section introduces practical batch and online implementations of the GPQ approach. Moreover, an implementation of the online GPQ, which makes use of the budgeted online sparse GP inference (see the section "Sparse Gaussian Processes for Online Settings" for more details) is also presented. Finally, we evaluate this method, showing that not only does it have useful theoretical properties, but it also results in competitive learning speeds.

In what follows, Q^* is the true optimal Q-function and $\hat{Q}^*$ is an agent's estimate of it, constructed based on the available data. In GPQ, this estimate will be modeled as a GP with mean function $m^*(\boldsymbol{x}, \boldsymbol{u})$ and positive-semidefinite covariance kernel $k([\boldsymbol{x}, \boldsymbol{u}], [\boldsymbol{x}', \boldsymbol{u}'])$. Assume a zero-mean Gaussian prior is placed on the distribution over the possible Q-functions, so that $Q(\boldsymbol{x}, \boldsymbol{u}) \sim \mathcal{N}(0, k(\cdot, \cdot))$. The objective of the algorithm is to perform posterior inference using available training data so that the current estimate of the mean $\hat{m}$ approaches the mean of Q^*. Let the current estimate of the mean of the Q-function be $\hat{Q}(\boldsymbol{x}, \boldsymbol{u}) = \hat{m}(\boldsymbol{x}, \boldsymbol{u})$. Since samples of the optimal Q^* are unavailable (since it is the function being learned), posterior inference must be performed using the best estimate of Q^* at the current time; specifically,

$$\hat{Q}(\boldsymbol{x}_t, \boldsymbol{u}_t) = r(\boldsymbol{x}_t, \boldsymbol{u}_t) + \gamma \max_{\boldsymbol{u}'} \hat{Q}(\boldsymbol{x}_{t+1}, \boldsymbol{u}'). \tag{45}$$

Note that (45) essentially provides a measurement model for generating GP samples: every time $\hat{Q}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is computed, $\{[\boldsymbol{x}_t, \boldsymbol{u}_t], \hat{Q}(\boldsymbol{x}_t, \boldsymbol{u}_t)\}$ becomes a new sample for updating the parameters of GPQ, including the set of basis vectors $\mathcal{VB}$ and GPR weight vectors $\boldsymbol{\alpha}$.

Batch Gaussian Process-Fitted Q-Iteration (GP-FQI)

Using GPs and the update rule in (45), the batch setting results in Algorithm 6 are referred to as GP-FQI, since they are members of the fitted Q-iteration family of algorithms [111]. Let T denote the approximate Bellman operator, that is, a procedure that updates the mean $\hat{m}$ of the current estimate of the Q-function using (45). Updates to the mean of the estimated Q-function are obtained by $\hat{m}_{t+1} = T\hat{m}_t$. It is known that T is a contraction mapping, so a fixed point exists [16]. For a GP model, the approximate Bellman operator in the batch case can be defined as training a new GP using the observations $\{y_i \,|\, y_i = r(\boldsymbol{x}_i, \boldsymbol{u}_i) + \gamma \max_{\boldsymbol{u}'} \hat{Q}(\boldsymbol{x}_i', \boldsymbol{u}')\}_{i=1}^N$

Convergence Analysis of Gaussian Process Q

The properties of Algorithm 6 can be summarized by the following theorems. The proofs are discussed in [102]. The following theorem shows that, in the case of finite data, a finite regularization term always exists that guarantees convergence.

Theorem S3

Given a Gaussian process (GP) with training data $\mathcal{Z}$ of finite size N and a Mercer kernel that is bounded above by $k_{\max}$, there exists a finite regularization parameter ω_n^2 such that the Bellman operator T is a contraction in the batch setting. In particular, $\omega_n^2 = 2(\|\boldsymbol{K}(\mathcal{Z},\mathcal{Z})\|_\infty - k_{\max}) \le 2N$. The crux of the proof is showing that the Bellman operator is a contraction if the term $\|\boldsymbol{K}(\mathcal{Z},\mathcal{Z})\|\,\|(K(\mathcal{Z},\mathcal{Z}) + \omega_n^2\boldsymbol{I})^{-1}\| \le 1$ when ω_n^2 is set in this way.

The next theorem shows that, for a GP with *infinite* training data (but only adds data points exceeding the linear independence test β_{tol}), a finite regularization term also exists.

Theorem S4

Given a GP with infinite data generated using a sparse approximation with acceptance tolerance β_{tol} and given a Mercer kernel function that decays exponentially, there exists a finite regularization parameter ω_n^2 such that the Bellman operator T is a contraction in the batch setting.

The key to the proof is that $\|\boldsymbol{K}(\mathcal{Z},\mathcal{Z})\| = \max_j \Sigma_i k(\mathbf{z}_j,\mathbf{z}_i)$, which is convergent for an infinite number of data points selected using the linear independence test in (14). Theorem S4 provides a powerful insight into the convergence properties of GPs in the context of the Bellman operator, central to control applications within the reinforcement learning setting. As the density of basis vectors increases or as the bandwidth of the kernel function grows (corresponding to decreasing β_{tol}), the basis vector weights α_i become increasingly correlated. As the weights become correlated, changing the weight of one basis vector also changes the weights of nearby basis vectors. It is this sharing of weights that can result in divergence, as shown in [99]. Theorem S4 shows that for a given β_{tol} and kernel function, there exists a finite regularization parameter ω_n^2 that will prevent divergence.

The next theorem shows that the approximation error from using a sparse representation of a GP versus a regular GP is bounded. The key to the proof is that the maximum approximation error is linear in β_{tol}.

Theorem S5

If the sparse GP algorithm is used, then the error $\|\mathbb{E}[\hat{Q} - Q^*]\|$ is uniformly, ultimately bounded for the Bellman operator.

Finally, Theorem S6 provides a set of sufficient conditions for convergence in the online case. The proof of Theorem S6 is based on similar techniques to [S8, Thrm 17]; in particular, it is based on using an ordinary differential equation representation of the $\boldsymbol{\alpha}$ update equation (19): $\dot{\boldsymbol{\alpha}}(t) = \mathbb{E}_\pi[\boldsymbol{q}_t \boldsymbol{S}_t]$, then showing that $\boldsymbol{\alpha} \to \boldsymbol{\alpha}^*$ for each active basis set.

Theorem S6

With an ergodic sampling policy π, for each active basis set $\mathcal{BV}$ defined in (16), a sufficient condition for convergence with probability one of $\hat{m}(\mathbf{z}_t) \to m^*(\mathbf{z}_t)$ as $t \to \infty$ when using the online sparse GP (Algorithm 2) with measurement model in (45) is $\mathbb{E}_\pi[\boldsymbol{C}_t\mathbf{k}_t\mathbf{k}_t^T + \boldsymbol{K}_t^{-1}\mathbf{k}_t\mathbf{k}_t^T] \ge \gamma\mathbb{E}_\pi[\boldsymbol{C}_t\mathbf{k}_t\mathbf{k}_t^\alpha + \boldsymbol{K}_t^{-1}\mathbf{k}_t\mathbf{k}_t^\alpha]$, where $\mathbf{k}^{\alpha_t}\boldsymbol{\alpha}_t = \max_{a'}(\mathbf{k}^T(\mathbf{x}_{t+1}, a'))\boldsymbol{\alpha}_t$.

REFERENCE

[S8] A. Benveniste, M. Métivier, and P. Priouret, *Adaptive Algorithms and Stochastic Approximations*, vol. 22. Berlin: Springer Science & Business Media, 2012.

at the input locations $\{z_i \,|\, z_i = (\boldsymbol{x}_i, \boldsymbol{u}_i)\}_{i=1}^N$. Intuitively, the Bellman operator uses the currently estimated Q-function to generate new measurements or observations via (45). It then updates the mean of the estimated Q-function based on the additional data, and the process is repeated. Since this operator is a contraction mapping, the update process eventually converges to a fixed-point estimate of Q^*.

Online Learning and Exploration with GPQ

GP-FQI is a method for learning the Q-function online with provable convergence. An ϵ-greedy exploration policy could be employed to guarantee sufficient data collection, and GP-FQI could perform batch updates after every step. However, because of the computational costs of this approach, an alternative approach is possible: *online GPQ*, presented in Algorithm 7. At each step of online GPQ, an action is taken according to a policy π that ensures ergodicity of the induced Markov chain, and the value $y_t = r_t + \gamma \max_{\boldsymbol{u}} \hat{Q}_\tau(\boldsymbol{x}_{t+1}, \boldsymbol{u})$ at location z_t is calculated. The GP sparsification technique (Algorithm 2) is used to determine whether or not to add a new basis vector (a state-action pair z_t) to the active bases set $\mathcal{BV}$ and the kernel weights are updated. This test ensures that only a sparse quantity of data/basis vectors will be kept in memory, which is important in the online setting. Note that adding a new basis vector to $\mathcal{BV}$ has the effect of expanding the space of Q-functions that can be represented by the GP, since the set of basis functions that spans that space is augmented. After this occurs, the GP parameters are updated according to Algorithm 7.

While ϵ-greedy exploration policies are sufficient to guarantee convergence of the algorithm, in practice they are often inefficient at gathering useful samples. A guiding principle of more sample-efficient RL algorithms [112] is "optimism in the face of uncertainty," where the process starts with an overestimation of the value function and selects greedy

actions; this is known to automatically balance exploration and exploitation. Alternatively, it is possible to use upper-confidence tails from the GP as an optimistic value function. Specifically, for any input point $\langle \boldsymbol{x}, \boldsymbol{u} \rangle$, the GP will report an upper-confidence tail of $m(\boldsymbol{x}) + 2\Sigma(\boldsymbol{x})$, where m and Σ are the posterior mean and variance given by (10) and (12). The online GPQ algorithm can then be modified to use greedy actions with respect to this upper tail; the update law associated with this algorithm is given by $\hat{Q}(\boldsymbol{x}_t, \boldsymbol{u}_t) = r(\boldsymbol{x}_t, \boldsymbol{u}_t) + \gamma \max_{\boldsymbol{u}'} [\hat{Q}(\boldsymbol{x}_t, \boldsymbol{u}') + 2\Sigma(\boldsymbol{x}_t, \boldsymbol{u}')]$. It implies that the upper tail of the next state's Q-value (in the Bellman update) is used to overestimate the value function; this is similar in nature to the motivating ideas of underlying techniques such as model-based interval estimation [112].

The rest of this section discusses experiments evaluating two variants of online GPQ: one with ϵ-greedy exploration and the other using optimistic value function initialization. Comparisons are made to Q-learning with linear function approximation and fixed bases (referred to as QL-FB); to GQ algorithm with fixed bases [105] and tabular Q-learning. Experiments are performed in two domains: a discrete 5×5 gridworld and a continuous-state inverted pendulum problem [114]. The gridworld consists of 25 cells in which an agent can navigate. The agent always starts in the lower-left corner and the reward function is zero per action executed, unless the agent is at the goal state (in the upper-right corner). In this case, it incurs a reward of one. In this experiment, the discount rate γ is 0.9. Transitions are noisy with a 0.1 probability of the agent staying in its current location independent of the action executed. There are five available actions, including movement in four directions and doing nothing. The inverted pendulum problem, on the other hand, consists of a continuous, two-dimensional state space with three actions: applying forces of –50, 0, or 50 N to a cart where a pendulum is attached. The state variables include the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. All three actions are corrupted by Gaussian noise with mean $\mu = 0$ N and standard deviation std = 10 N. The state transitions are governed by the nonlinear dynamics of the system and depend on the current state $[\theta, \dot{\theta}]$ and current control input u

$$\ddot{\theta} = \frac{g \sin(\theta) - \rho m_p L (\dot{\theta}) \sin(2\theta)/2 - \rho \cos(\theta) u}{4L/3 - \rho m_p L \cos^2(\theta)}, \tag{46}$$

where $m_p = 2.0$ kg is the mass of the pendulum, $g = 9.8$ m/s^2 is the gravity constant, $M = 8.0$ kg is the mass of the cart, $L = 0.5$ m is the length of the pendulum, and $\rho = 1/(m_p + M)$. The frequency of applying the control input is 10 Hz. The goal is to balance the pendulum upright within a target angle in the interval of $(-\pi/2, \pi/2)$ rad. The reward at a given state is defined as the difference between the absolute value of the angle of the pendulum for two consecutive states.

Results are influenced by several algorithm hyper-parameters, including: 1) the learning rate (affecting QL-tabular, QL-FB, and GQ), 2) the exploration rate (affecting QL-tabular, QL-FB, GQ, and GPQ-ϵ-greedy), 3) the bandwidth of the kernel function (affecting QL-FB, GQ, and GPQ), 4) the position of kernels (affecting GQ, QL-FB, GPQ), and 5) the number of kernels (that is, the quantization level in QL-tabular). The learning rate is set as $0.5/t^{\alpha}$, where t is the number of interactions within the environment and $\alpha \in \{0.1, 0.3, 0.5\}$, and the exploration rate is set according to $1/t^{\beta}$, with $\beta \in \{0.1, 0.3, 0.5\}$. The gridworld problem uses a budget of 25 kernel vectors, each one corresponding to a vector $[k(\boldsymbol{c}_i, \boldsymbol{c}_1), \ldots, k(\boldsymbol{c}_i, \boldsymbol{c}_{25})]^T$; that is, a vector whose elements are defined by evaluating the kernel function k between one particular center location $\boldsymbol{c}_i$ in the gridworld and each of the other 25 possible centers. For the inverted pendulum problem, the budget is selected from the set {36, 100}. The quantization level used to discretize this continuous problem (so that it can be solved via tabular Q-learning) is set as the budget for the function approximation methods.

After parameter tuning and cross-validation experiments, the policy learned by these methods (on both domains) was evaluated based on the discounted cumulative reward they achieved, averaged over 20 independent runs. For each of the algorithms, the best combination of parameter

ALGORITHM 6 Batch Gaussian process-fitted Q-iteration (GP-FQI).

Input: Experience tuples $\langle \boldsymbol{x}_i, \boldsymbol{u}_i, r_i, \boldsymbol{x}_i' \rangle_{i=1\ldots N}$
Output: A GP representing $\hat{Q}^*$
$\hat{Q} \leftarrow$ Initialized GP
repeat
  $\hat{Q}' \leftarrow$ Initialized GP
  for each experience tuple $\langle \boldsymbol{x}_i, \boldsymbol{u}_i, r_i, \boldsymbol{x}_i' \rangle$ **do**
    $y_i = r_i + \gamma \max_{\boldsymbol{b}} \hat{Q}(\boldsymbol{x}_i', \boldsymbol{b})$
  end for
  Train $\hat{Q}$ on all $(\langle \boldsymbol{x}_i, \boldsymbol{u}_i \rangle, y_i)$
  $\hat{Q} = \hat{Q}'$
until the convergence condition is satisfied

ALGORITHM 7 Online Gaussian process Q-learning.

for for each time step τ **do**
  Choose $\boldsymbol{u}_\tau$ from $\boldsymbol{x}_\tau$, using ϵ-greedy exploration
  Take action $\boldsymbol{u}_\tau$, observe $r_\tau, \boldsymbol{x}_{\tau+1}$
  Let $\boldsymbol{z}_\tau = \langle \boldsymbol{x}, \boldsymbol{u} \rangle$ and $y_\tau = r_\tau + \gamma \max_{\boldsymbol{b}} \hat{Q}(\boldsymbol{x}_{\tau+1}, \boldsymbol{b})$
  if $\beta_{\tau+1} > \beta_{\text{tol}}$ **then**
    Add $\boldsymbol{z}_\tau$ to the $\mathcal{BV}$ set.
  end if
  Compute $\mathbf{k}_{\boldsymbol{z}_{\tau+1}}$ and $\boldsymbol{\alpha}_{\tau+1}$ according to (11)
  if $|\mathcal{BV}| >$ Budget **then**
    Delete $\boldsymbol{z}_i \in \mathcal{BV}$ with lowest score according to (24)–(27)
  end if
  update $\hat{Q}(\boldsymbol{z}_{\tau+1}) = \sum_{i=1}^{\infty} \alpha_i k(\boldsymbol{z}_i, \cdot)$
end for

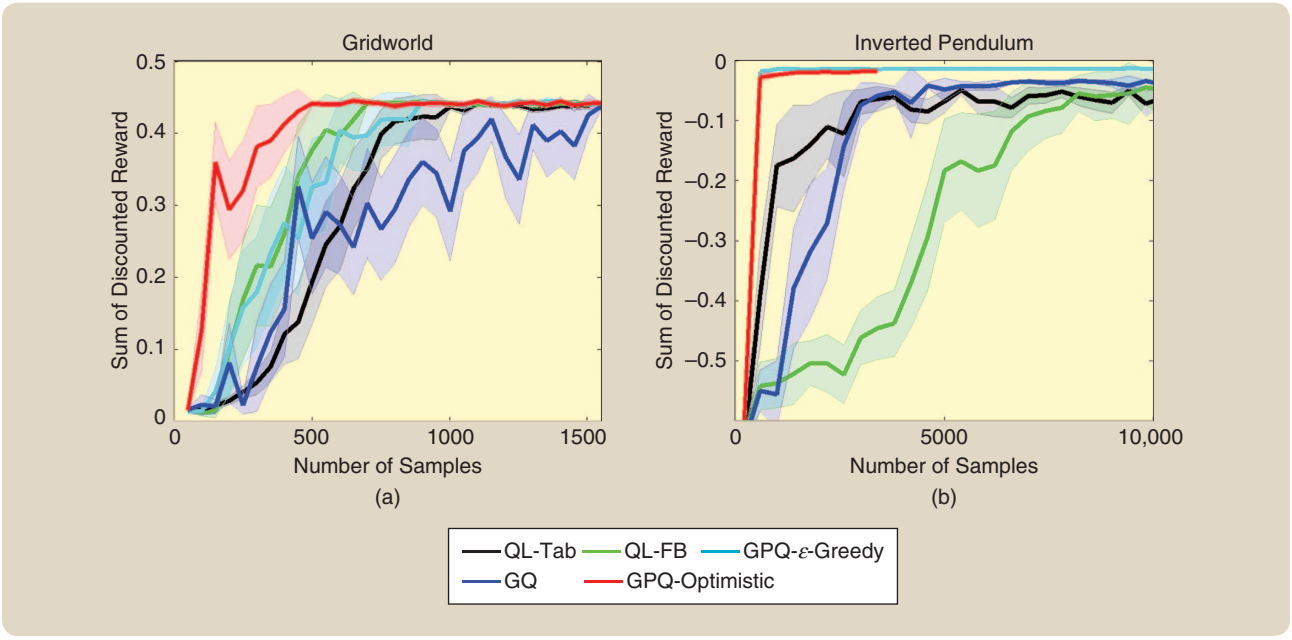

FIGURE 10 The average sum of discounted rewards for two domains: (a) Gridworld and (b) the inverted pendulum. The GQ algorithm and quantization-level (QL) variants are given more domain knowledge since a designer needs to manually specify the bases functions to be used. Even though Gaussian process (GP) Q automatically selects which bases to use, in a data-driven manner, it is nonetheless capable to reaching comparable performance (often faster). (Figure reproduced with permission from [102] and with code available in [113].)

settings was used, and the corresponding results are shown in Figure 10.

While all of the algorithms find the optimal policy in the gridworld experiments, the GPQ-based methods converge much faster by quickly identifying important areas for basis points. Optimistic exploration using the GP's variance is advantageous, as the algorithm very quickly uncovers the optimal policy. In inverted pendulum, GPQ quickly finds adequate bases and converges to a near-optimal policy, while GQ requires more samples. Beyond the "best-parameter" results shown in this graph, the GPQ methods are observed to be more resilient against small quantizations (budgets) (because they are able to select their own bases), while GQ and QL-FB are far more sensitive to the number and placement of bases.

The following experiments from the inverted pendulum domain show the robustness of GPQ against changes in the quantization (budgeting) level and the sensitivity of the fixed-basis methods to the number of basis points. Figure 11 demonstrates that GPQ methods are more resilient against small budgets (because they are able to select their own bases), while GPQ and QL-FB are more volatile. These graphs also show that optimistic exploration is beneficial for certain parameter settings.

APPLICATION IV: INVERSE OPTIMAL CONTROL WITH GAUSSIAN PROCESSES

Inverse optimal control refers to the problem of identifying or inferring, from a set of expert demonstrations, the unknown reward or cost function of a decision process, such as an MDP. This plays a significant role in problems such as programming robots through demonstrations, since such functions often characterize the implicit goals of the experts generating the demonstrations. They are also succinct, robust, and transferable definitions of a control task [115]. Inverse optimal control was originally established for linear systems by Kalman [116] and solved in [117]. More recently, this problem was studied within the RL framework [115] as IRL, highlighting that a reward function is not known a priori and must instead be learned in some applications. This section discusses how GPs can be used to model unknown reward functions in some RL applications to solve IRL problems, particularly in cases where using linear models to represent the unknown reward function results in inefficient or inaccurate representations.

In an IRL problem, all components of an MDP $\mathcal{M}$ are known or specified except its reward function r: $\mathcal{M}/r$. An IRL algorithm is then presented with observations (expert demonstrations), denoted $\mathcal{O} = \{\xi_1, \ldots, \xi_N\}$, where ξ_n is a trajectory of state-action pairs $\xi_n = \{(\boldsymbol{x}_{n,0}, \boldsymbol{u}_{n,0}), \ldots (\boldsymbol{x}_{n,T_n}, \boldsymbol{u}_{n,T_n})\}$. IRL also assumes a given reward feature representation $\phi : \mathcal{X} \to \mathbb{R}$, which are linearly combined to form an estimate of the unknown reward function. The objective of an IRL algorithm is to recover a reward function consistent with the observed demonstrations, that is, a reward function $\hat{r}(\boldsymbol{x})$ whose corresponding optimal policy π^* matches the observations $\mathcal{O}$. Based on the Bellman optimality equations

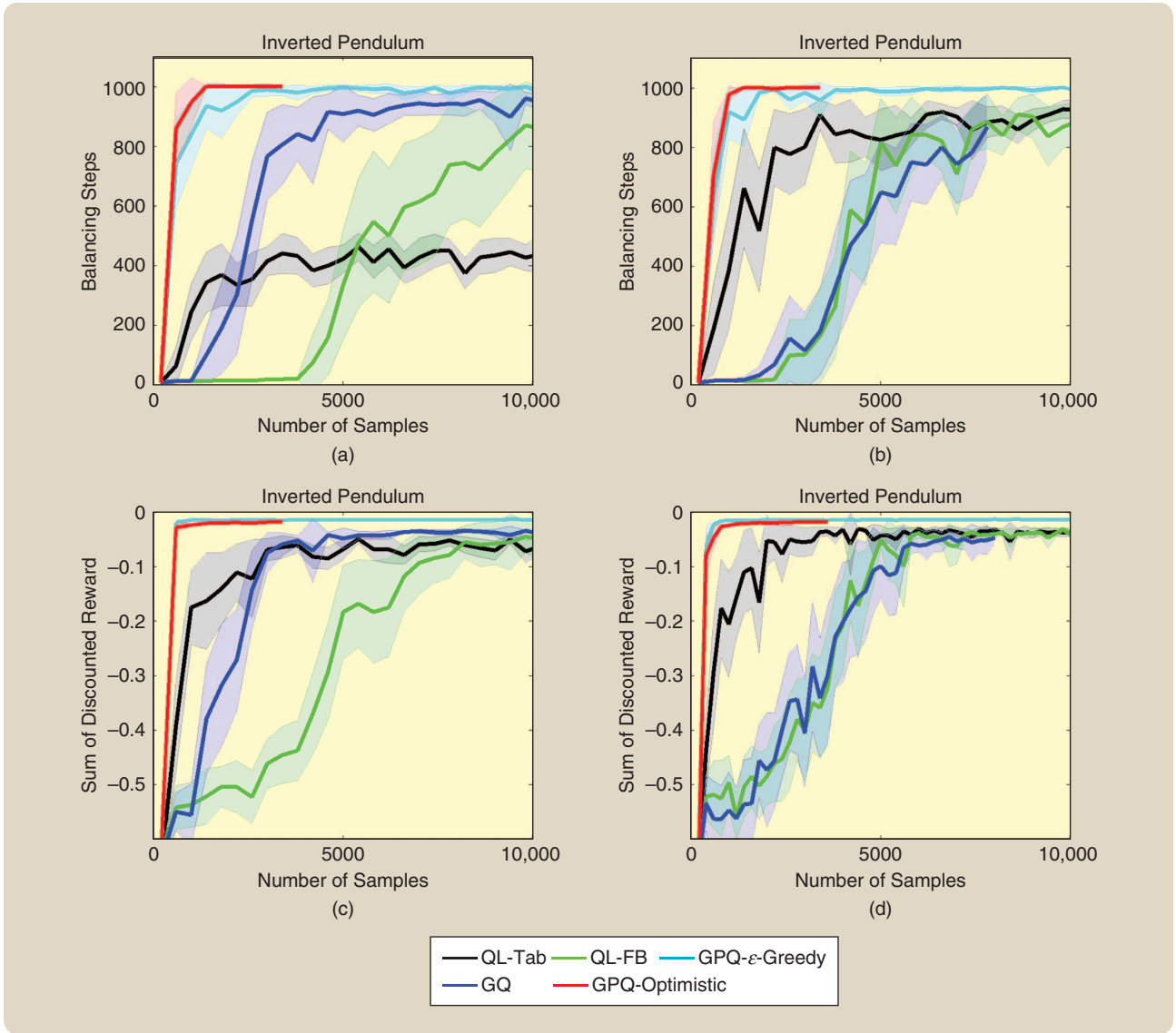

FIGURE 11 The performance of the inverted pendulum under different budgets (quantization levels): 36 (left) and 100 (right). Gaussian process (GP) Q is insensitive to the quantization level, because it selects its own bases, while the quantization level significantly impacts the other algorithms. For example, with a higher quantization-level (QL) GPQ converges slowly, and with a lower quantization level, QL-tab performs poorly. (Figure reproduced with permission from [102] and with code available at [113].)

(39) for the discrete-state case, it can be shown that the only reward vectors $\boldsymbol{r} \in \mathbb{R}^{|\mathcal{X}|}$ consistent with π^* are those satisfying the inequality

$$(\boldsymbol{P}^{\pi^*} - \boldsymbol{P}^{\pi})(\boldsymbol{I} - \gamma \boldsymbol{P}^{\pi^*})^{-1}\boldsymbol{r} \geq \boldsymbol{0}, \tag{47}$$

where $\boldsymbol{P}^{\pi} \in \mathbb{R}^{|\mathcal{X}|\,|\mathcal{A}| \times |\mathcal{X}|\,|\mathcal{A}|}$ is the transition probability for the state-action pairs with $P^{\pi}(\boldsymbol{x}', \boldsymbol{u}' \,|\, \boldsymbol{x}, \boldsymbol{u}) = P(\boldsymbol{x}' \,|\, \boldsymbol{x}, \boldsymbol{u})\pi(\boldsymbol{x}' \,|\, \boldsymbol{u}')$. Note that although this assumes a discrete state space, a similar inequality also exists for the case of continuous state spaces; we do not present it here to simplify the notation.

Although IRL has a well-defined objective, recovering the reward function is an ill-posed problem, since the same behavior (set of expert demonstrations) can be explained by multiple reward functions. Moreover, human demonstrations are usually not optimal, hence directly reproducing the observed behavior of a suboptimal expert may be inappropriate. Instead, it might be preferable to assume a probabilistic model over the possible behaviors that can be generated by an expert so that the reward functions identified by IRL are computed based on all available demonstrations—both optimal and suboptimal. One popular approach to learn from a suboptimal expert and address the multiple reward functions that may be consistent with the demonstrations is to use a probabilistic model of the expert's policies. This can be achieved by using a *maximum causal entropy* IRL model [118]. Under this model, the probability of a trajectory

To spark the control community's interest in GP-related techniques, this tutorial uses control and RL applications as motivating examples and shows how GPs can model key functions underpinning many control problems.

ξ is proportional to an exponential function of the rewards $\boldsymbol{r}$ associated with that trajectory. This can be achieved by assuming that the policy is a probabilistic model $\pi(\boldsymbol{u}\,|\,\boldsymbol{x}) \propto \exp Q^r(\boldsymbol{x},\boldsymbol{u})$, where $Q^r(\boldsymbol{x},\boldsymbol{u}) = r(\boldsymbol{x},\boldsymbol{u}) + \gamma T V^r(\boldsymbol{x})$ with $V^r(\boldsymbol{x}) = \log \Sigma_u \exp Q^r(\boldsymbol{x},\boldsymbol{u})$ being defined as a value function. As a result of these assumptions, the action selection probability $\pi(\boldsymbol{u}\,|\,\boldsymbol{x})$ can be obtained by applying a normalization factor $\exp V^r(\boldsymbol{x})$, thus leading to $\pi(\boldsymbol{u}\,|\,\boldsymbol{x}) = \exp(Q^r(\boldsymbol{x},\boldsymbol{u}) - V^r(\boldsymbol{x}))$. Note that when the differences between all actions' Q-values reach the same level, the probabilities of those actions under the policy become uniform. Similarly, when one specific action's Q-value dominates all others, then the stochastic maximum causal entropy policy becomes deterministic and matches the optimal policy [118]. Note that using the maximum entropy method to solve IRL requires optimizing the following likelihood of a reward model $\boldsymbol{r}$ given trajectories $\mathcal{O}$:

$$p(\mathcal{O}\,|\,\boldsymbol{r}) = \prod_{n=1}^{N}\prod_{t=0}^{T_n} \pi(\boldsymbol{u}_{n,t}\,|\,\boldsymbol{x}_{n,t})\, p(\boldsymbol{x}_{n,t+1}\,|\,\boldsymbol{x}_{n,t},\boldsymbol{u}_{n,t}) \propto \exp\left\{\sum_{n=1}^{N}\sum_{t=0}^{T_n}(Q^r(\boldsymbol{x}_{n,t},\boldsymbol{u}_{n,t}) - V^r(\boldsymbol{x}_{n,t}))\right\}. \tag{48}$$

Although $\boldsymbol{r}$ can be estimated by directly maximizing (48), such a reward function is only defined for states that are part of observed trajectories but does not generalize to unseen states. To address this issue, many existing IRL methods represent the reward function as a linear function

$$r(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}), \tag{49}$$

where $\boldsymbol{w}$ is a vector of weights and $\boldsymbol{\phi}$ is a set of (given) reward features. However, if $\boldsymbol{r}$ is nonlinear in the feature or the features are not provided, this linear model is not sufficiently expressive.

More recently, GP-based IRL methods (collectively called GPIRL) [119]–[121] were developed to combine probabilistic reasoning about stochastic expert behavior and the ability to learn a nonlinear reward function. GPIRL has been shown to outperform many existing methods on tasks with inherently nonlinear rewards and suboptimal examples. Let $\boldsymbol{g}$ be the (unknown) set of rewards associated with states $\mathcal{X}_g$ visited in the expert demonstrations $\{\xi_1,\ldots,\xi_N\}$. Instead of directly identifying $\boldsymbol{r}$ (the complete reward function over the entire state space $\mathcal{X}$), GPIRL methods first estimate $\boldsymbol{g}$ and then, based on this estimate, infer how it may be generalized to the rest of the (unseen) state space. The density of the rewards $\boldsymbol{g}$ under the expert demonstration $\mathcal{O}$ is given by

$$p(\boldsymbol{g}\,|\,\mathcal{O},\mathcal{X}_g,\mathcal{H}_p) \propto p(\mathcal{O},\boldsymbol{g}\,|\,\mathcal{X}_g,\mathcal{H}_p) = \left[\int_r p(\mathcal{O}\,|\,\boldsymbol{r})\, p(\boldsymbol{r}\,|\,\boldsymbol{g},\mathcal{X}_g)\, d\boldsymbol{r}\right] p(\boldsymbol{g}\,|\,\mathcal{X}_g,\mathcal{H}_p), \tag{50}$$

where $p(\mathcal{O}\,|\,\boldsymbol{r})$ is given by (48), $p(\boldsymbol{g}\,|\,\mathcal{X}_g,\mathcal{H}_p) = \mathcal{N}(\boldsymbol{0},\boldsymbol{K}_{g,g})$ is the GP prior, and $p(\boldsymbol{r}\,|\,\boldsymbol{g},\boldsymbol{X})$ is the density of the complete reward function (that is, the reward over all states, not only at the ones in the observed trajectories) under current values of $\boldsymbol{g}$ (GP posterior). It can be shown that $p(\boldsymbol{r}\,|\,\boldsymbol{g},\boldsymbol{X})$ is a Gaussian with mean $\boldsymbol{K}_{r,g}^T \boldsymbol{K}_{g,g}^{-1}\boldsymbol{g}$ and covariance function $\boldsymbol{K}_{r,r} - \boldsymbol{K}_{r,g}^T \boldsymbol{K}_{r,r}^{-1}\boldsymbol{K}_{r,g}$, where $\boldsymbol{K}_{r,g}$ is the covariance between rewards at all states in $\mathcal{X}$ and rewards at states in $\mathcal{X}_g$. Because $p(\mathcal{O}\,|\,\boldsymbol{r})$ has a complicated form, the integral in (50) does not have a closed-form solution. Solving this equation can, however, be seen as a problem analogous to the sparse approximation for GPR, where only a small set of basis vectors $\boldsymbol{g}$ acts as the set of basis vectors spanning the complete set of points $\boldsymbol{r}$. Under this interpretation, $\boldsymbol{r}$ is set to $\boldsymbol{r} = \boldsymbol{K}_{r,g}^T \boldsymbol{K}_{g,g}^{-1}\boldsymbol{g}$, and the density in (50) becomes

$$p(\mathcal{O},\boldsymbol{g}\,|\,\boldsymbol{X},\mathcal{H}_p) = p(\mathcal{O}\,|\,\boldsymbol{r} = \boldsymbol{K}_{r,g}^T \boldsymbol{K}_{g,g}^{-1}\boldsymbol{g})\, p(\boldsymbol{g}\,|\,\mathcal{X}_g,\mathcal{H}_p), \tag{51}$$

where $\mathcal{H}_p$ is the set of kernel hyperparameters. After optimizing the above likelihood, the complete reward function $\boldsymbol{r}$ can be inferred via $\boldsymbol{r} = \boldsymbol{K}_{r,g}^T \boldsymbol{K}_{g,g}^{-1}\boldsymbol{g}$. This reward function can then be subsequently used to recover the expert's policy on the entire state space, as opposed to only on states that are part of the observed trajectories, via algorithms such as GP-DP [see the section "Application II: Planning and Optimal Control—Gaussian Process Dynamic Programming (GP-DP) and Model Predictive Control (MPC)"]. In problems with nonlinear reward functions [119], GPIRL was shown to outperform many existing methods. Moreover, when presented with unseen states, GPIRL is capable of estimating their corresponding rewards based on the posterior mean, thereby demonstrating a key benefit of GPs in this setting, namely, accounting for uncertainty about rewards in states that are different from those in the demonstrations. Note that such an uncertainty also provides useful information to the learning algorithm, since it allows a designer to inform the expert about states (possibly

with high reward variance) that may require additional demonstrations. GPIRL was evaluated not only on simulated tasks but also on learning challenging maneuvers from demonstrations on a quadrotor helicopter and radio-controlled car [122].

GAUSSIAN PROCESS REGRESSION AND ITS CONNECTIONS TO OTHER METHODS

The previous sections introduced the basics of GPR and demonstrated its representational power and usefulness for solving a broad spectrum of control-related problems. To highlight the unique features of GPs and their generality, this section further discusses the connections between GPR methods and other well-known methods in statistics and machine learning. This section also briefly introduces state-of-the-art GP-based methods that may be applied for solving challenging control problems, and it discusses a few limitations of GP models. Finally, this section presents novel technical developments that extend the basic GP model and can be used to address such limitations.

Gaussian Processes and Statistical Methods

Many methods have been proposed by the statistics community for use in control problems; some of these are closely related to GPR. This section highlights and discusses three well-known methods of this type (see [26] and [48] for a comprehensive discussion).

Gaussian Process Regression as a Linear Smoother

A linear smoother is a regression function defined as the average of its training outputs:

$$\hat{y}(\boldsymbol{z}_*) = \sum_{i=1}^{N} \alpha_i(\boldsymbol{z}_*) y_i, \tag{52}$$

where $\boldsymbol{z}_*$ is a new input location and the set $\{y_i\}$ consists of the observed output values in the training set. To understand why GPR is a linear smoother, the mean of the posterior predictive distribution of a GP (10) is rewritten as

$$\bar{g}(\boldsymbol{z}_*) = \boldsymbol{k}_*^T (\boldsymbol{K} + \omega_n^2 \boldsymbol{I})^{-1} \boldsymbol{y} = \sum_{i=1}^{N} y_i \alpha_i(\boldsymbol{z}_*), \tag{53}$$

where each weight $\alpha_i(\boldsymbol{x}_*)$ corresponds to the ith element of the kernel weight vector $\boldsymbol{\alpha} = (\boldsymbol{K} + \omega_n^2 \boldsymbol{I})^{-1} \boldsymbol{k}_*$. Moreover, considering the eigendecomposition of $\boldsymbol{K}$ (which is $\boldsymbol{K} = \Sigma_{i=1}^{N} \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^T$), since $\boldsymbol{K}$ is a positive-definite matrix, the eigenvalues $\{\lambda_i\}_{i=1}^{N}$ are real and nonnegative, and all eigenvectors $\{\boldsymbol{e}_i\}_{i=1}^{N}$ are orthonormal. Let $\boldsymbol{y} = \Sigma_{i=1}^{N} \gamma_i \boldsymbol{e}_i$, where $\gamma_i = \boldsymbol{e}_i^T \boldsymbol{y}$. The prediction of the input from the training set is

$$\boldsymbol{f} = \boldsymbol{K}(\boldsymbol{K} + w_n^2 \boldsymbol{I})^{-1} \boldsymbol{y} = \sum_{i=1}^{N} \frac{\gamma_i \lambda_i}{\lambda_i + w_n^2} \boldsymbol{e}_i. \tag{54}$$

Note that if $\lambda_i / (\lambda_i + w_n^2) \ll 1$, the component in $\boldsymbol{y}$ along the corresponding basis $\boldsymbol{e}_i$ is effectively filtered out. For most widely used kernel covariance functions, the eigenvalues corresponding to slow-varying eigenvectors are large. Therefore, high-frequency components in $\boldsymbol{y}$ are smoothed out.

Note that in the case of linear smoothers, the weight function is derived from a smoothing rather than a Mercer kernel. While a smoothing kernel only has local support and its weights must sum to one (that is, $\Sigma_{i=1}^{N} \alpha_i(\boldsymbol{z}_*) = 1$), a Mercer kernel can be nonlocal. As a consequence, the effective bandwidth of the equivalent kernel of a GP automatically decreases as the sample size N increases. In kernel smoothing, the bandwidth must be set by hand to adapt to N (see [26, Sec. 2.6 and Sec. 7.1] for details).

Neural Networks as Gaussian Processes

Consider an NN for regression with one hidden layer that uses the following model

$$p(y \,|\, \boldsymbol{z}, \boldsymbol{\theta}) = \mathcal{N}(y \,|\, g(\boldsymbol{z}; \boldsymbol{\theta}), \sigma^2), \tag{55}$$

where $g(\boldsymbol{z}) = b + \Sigma_{j=1}^{H} v_j h(\boldsymbol{z}; \boldsymbol{u}_j)$, b is a bias term, v_j is the output weight of the hidden unit j to the response y, $\boldsymbol{u}_j$ are the input weights to unit j from the input layer $\boldsymbol{z}$, and $h(\cdot)$ is the activation function, typically the sigmoid or hyperbolic tangent function.

Given the prior distribution over weights $b \sim \mathcal{N}(0, \sigma_b^2)$, $\boldsymbol{v} \sim \Pi_j \mathcal{N}(v_j \,|\, 0, \sigma_v^2)$, and the distribution of hidden units $\boldsymbol{u} \sim \Pi_j p(\boldsymbol{u}_j)$ for an arbitrary probability distribution $p(\boldsymbol{u}_j)$. Denoting all of the weights by $\boldsymbol{\theta}$ yields

$$\mathbb{E}_{\boldsymbol{\theta}}[g(\boldsymbol{z})] = 0, \tag{56}$$

$$\begin{aligned} \mathbb{E}_{\boldsymbol{\theta}}[g(\boldsymbol{z}) g(\boldsymbol{z}')] &= \sigma_b^2 + \sum_{j=1}^{H} \sigma_v^2 \mathbb{E}_{\boldsymbol{u}}[h(\boldsymbol{z}; \boldsymbol{u}_j) h(\boldsymbol{z}'; \boldsymbol{u}_j)], \\ &= \sigma_b^2 + H \sigma_v^2 \mathbb{E}_{\boldsymbol{u}}[h(\boldsymbol{z}; \boldsymbol{u}) h(\boldsymbol{z}'; \boldsymbol{u})], \end{aligned} \tag{57}$$

where the last equality follows, because the hidden units $\{h(\boldsymbol{z}; \boldsymbol{u}_j)\}_{j=1}^{H}$ are independent and identically distributed. If σ_v^2 scales as w_n^2 / H, then the last term becomes $w_n^2 \mathbb{E}_{\boldsymbol{u}}[h(\boldsymbol{z}; \boldsymbol{u}) h(\boldsymbol{z}'; \boldsymbol{u})]$. If h is bounded, then applying the central limit theorem (as $H \to \infty$) yields a GP.

As an example, considering the activation function $h(\boldsymbol{z}; \boldsymbol{u}) = \operatorname{erf}(u_0 + \Sigma_{j=1}^{D} u_j z_j)$, where $\operatorname{erf}(a) = 2/\sqrt{\pi} \int_0^a e^{-t^2} dt$, and choosing $\boldsymbol{u} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$, the following covariance kernel function is obtained:

$$K_{NN}(\boldsymbol{z}, \boldsymbol{z}') = \frac{2}{\pi} \sin^{-1}\left(\frac{2 \tilde{\boldsymbol{z}}^T \boldsymbol{\Sigma} \tilde{\boldsymbol{z}}'}{\sqrt{(1 + 2 \tilde{\boldsymbol{z}}^T \boldsymbol{\Sigma} \tilde{\boldsymbol{z}})(1 + 2 \tilde{\boldsymbol{z}}'^T \boldsymbol{\Sigma} \tilde{\boldsymbol{z}}')}} \right), \tag{58}$$

which is an NN kernel with $\tilde{\boldsymbol{z}} = (1, z_1, \ldots, z_D)$.

Moreover, if $\boldsymbol{u} \sim \mathcal{N}(\boldsymbol{0}, \sigma_u^2 \boldsymbol{I})$, then it can be shown that using an RBF network (see "Limitations of Parametric Models—A Flight Control Example" for more details) is equivalent to using a hidden unit activation function of the form $h(\boldsymbol{z}; \boldsymbol{u}) = \exp(-|\boldsymbol{z} - \boldsymbol{u}|^2 / (2\sigma_h^2))$.

One limitation of the standard formulation of GPs is that the outputs must be scalar valued (although inputs can be multidimensional).

Linear Regression with a Reproducing Kernel Hilbert Space Regularizer Is Equivalent to MAP Estimation with a Gaussian Process

GPs were previously discussed as distributions over functions in some space $\mathcal{H}$ of interest. This statement can be made more precise by observing that under GPR, the mean of the process is assumed to lie in a class of functions $\mathcal{H}$ that is an RKHS [26]: a space in which functions $g \in \mathcal{H}$ satisfy $\|g\|_{\mathcal{H}} < \infty$, such that $\|g(\cdot)\|^2_{\mathcal{H}} = \Sigma_{i=1}^{\infty}\Sigma_{j=1}^{\infty}\alpha_i\alpha_j k(z_i, z_j)$. According to Mercer's theorem, any positive-definite kernel function can be represented in terms of eigenfunctions

$$k(z, z') = \sum_{i=1}^{\infty} \lambda_i \psi_i(z)\psi_i(z'), \tag{59}$$

where ψ_i represents an orthonormal basis for a function space

$$\mathcal{H} = \left\{ g : g(z) = \sum_{i=1}^{N} \alpha_i \psi_i(z), \sum_{i=1}^{\infty} \alpha_i^2/\lambda_i \right\}. \tag{60}$$

Let the inner product between two functions $g(z)$ and $f(z) = \Sigma_{i=1}^{\infty}\beta_i\psi_i(z)$ in space $\mathcal{H}$ be

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^{\infty} \alpha_i\beta_i/\lambda_i, \tag{61}$$

and consider an optimization (minimization) problem of the following form

$$C(g) = \frac{1}{2\omega_n^2}\sum_{i=1}^{N}(y_i - g(z_i))^2 + \frac{1}{2}\|g\|^2_{\mathcal{H}}, \tag{62}$$

where $\|g\|^2_{\mathcal{H}} = \langle g, g \rangle_{\mathcal{H}} = \Sigma_{i=1}^{\infty}(\alpha_i^2/\lambda_i)$ is the norm of function g. According to the representer theorem [26], the solution to the optimization problem must be in the form

$$g(z) = \sum_{i=1}^{N} a_i k(z, z_i). \tag{63}$$

Substituting (63) into (62) yields

$$C(\boldsymbol{a}) = \frac{1}{2w_n^2}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{a})^T(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{a}) + \frac{1}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}, \tag{64}$$

whose minimizer with respect to $\boldsymbol{a}$ is $\hat{\boldsymbol{a}} = (\boldsymbol{K} + w_n^2\boldsymbol{I})^{-1}\boldsymbol{y}$. Then,

$$g(z_*) = \sum_i \hat{a}_i k(z_*, z_i) = \boldsymbol{k}_*^T(\boldsymbol{K} + w_n^2\boldsymbol{I})^{-1}\boldsymbol{y}, \tag{65}$$

which is identical to the mean of the GP predictive distribution (10). Since the mode and mean of a Gaussian are the same, one can conclude that linear regression with an RKHS regularizer is equivalent to posterior estimation with a GP.

FURTHER ISSUES AND FUTURE DIRECTIONS IN GAUSSIAN PROCESS RESEARCH

Limitations of Basic Gaussian Process Regression Methods

The previous sections described a basic GP model and its applications. Under the assumption that measurement noise is Gaussian, the posterior of a GP can be derived in an analytic form. This assumption does not preclude the feasibility of performing a posterior inference of GPs with other likelihood functions, which, in general, may not have closed forms. Existing GPR algorithms also assume that a single generative model is used, which can lead to poor predictive performance when the data being modeled are nonstationary (when the data are generated by multiple switching processes). Inference over dynamic and nonstationary functions is a known limitation of the basic GP model. More advanced models exist that extend the basic GP model introduced in earlier sections. The following sections summarize the basic ideas underlying some of these methods.

Vector-Valued, or Dependent, Gaussian Processes

One limitation of the standard formulation of GPs is that the outputs must be scalar valued (although inputs can be multidimensional). A straightforward approach for allowing the use of GPs in problems requiring vector-valued outputs is to utilize multiple GPs, one for each output dimension. For example, in [123], two GPs were used to model the velocity fields of pedestrians for collision-free path planning. Using multiple GPs to model vector-valued processes, however, has the inherent limitation that the GP models are independent—information used to train one model is ignored by other models, even if they are correlated. Vector-valued GPs, or dependent GPs, have been designed to overcome this limitation [68], [69].

Figure 12 illustrates the difference between dependent and independent GPs. In this example, the goal is to model two one-dimensional time series, $f(t)$ and $g(t)$, given four training points. Among these training points, three correspond to measurements of f (blue dots) and one corresponds to a measurement of g (red dot). Figure 12(a) shows

Information value functions allow robots to actively decide which measurements to collect to gain the most information about a system with multiple dynamic patterns.

the resulting GPs obtained by modeling f and g independently, and Figure 12(b) shows the result obtained by modeling f and g as two outputs of a two-dimensional dependent GP. Solid lines represent the means of the processes, and colored bands represent prediction uncertainties. Note that the uncertainty of g in Figure 12(b) is much smaller than that in Figure 12(a). In particular, this occurs because a dependent GP model can exploit the dependency between f and g, thereby using three data points sampled from f to infer the output values of g. Finally, note that this dependency is bidirectional; hence, the only data point sampled from g also improves the estimation of f, which can be observed by the fact that the uncertainty decreases. This example demonstrates an important capability of dependent GPs that is missing from the independent GP models: the capability of sharing (in a principled way) information between different correlated channels.

Mixture of Gaussian Processes

For many tasks, such as modeling and tracking targets that are subject to nonlinear, switching dynamic behaviors, a single GP model might not be powerful enough—it might not be capable of capturing all of the different types of motion patterns and environment changes. Hence, GP mixture models [125] have been developed that allow the system to distinguish between multiple motion patterns. Finite-mixture models, however, are limited in flexibility since the number of motion patterns must be specified a priori. BNPMs, such as the Dirichlet process mixture of the Gaussian process model (DPGP) [126], address these limitations and allow the target dynamics (as well as the number of parameters) to be learned from data sampled continuously over time. As a result, the target model and posterior inference algorithms can be updated incrementally, without requiring any prior training or user interventions. DPGP models have been effectively used for modeling the dynamics of multiple targets motion patterns with a shared state space. Sample application domains include modeling traffic patterns [3], pedestrian tracking and avoidance [123], and learning robot dynamics for safe navigation under changing operating conditions [127]. More recently, *information value functions* (that is, value functions constructed based on rewards that measure information gain) have been proposed and used in conjunction with DPGP models

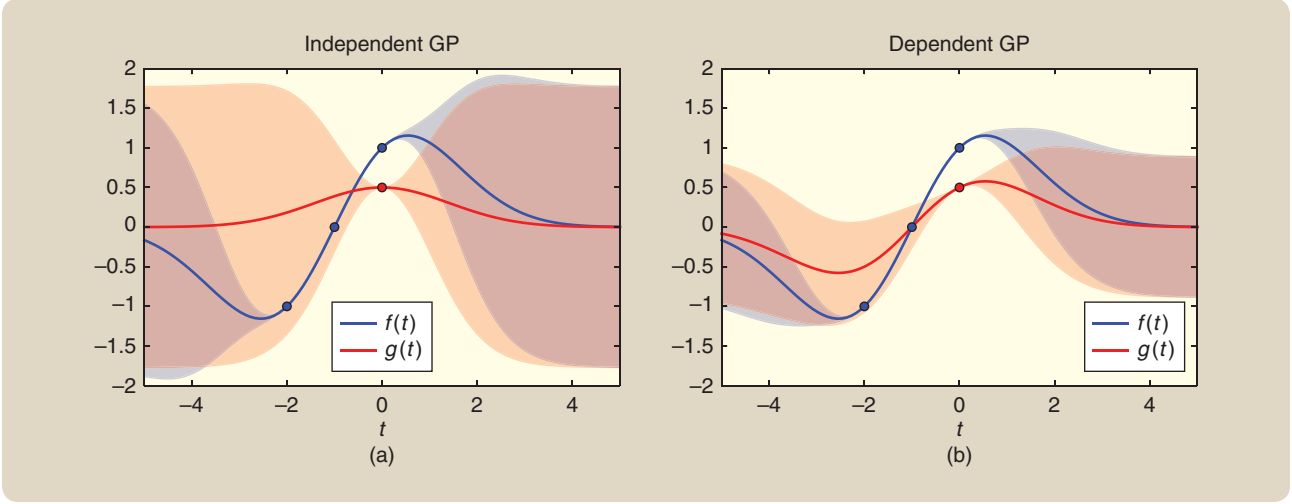

FIGURE 12 The difference between independent and dependent Gaussian processes (GPs). The goal is to model two one-dimensional time series, $f(t)$ and $g(t)$, given four training points. Among the training points, three correspond to measurements of f (blue dots), and one corresponds to a measurement of g (red dot). (a) shows the resulting GPs obtained by modeling f and g independently, and (b) shows the result obtained by modeling f and g as two outputs of a two-dimensional, dependent GP. Solid lines represent the means of the processes, and colored bands represent prediction uncertainties. The uncertainty of g in (b) is smaller than that in (a) because a dependent GP model exploits the dependency between f and g, using three points from f to infer the output values of g. (Figures generated using code available in [124].)

to derive optimal sensor measurement plans. In particular, information value functions allow robots to actively decide which measurements to collect to gain the most information about a system with multiple dynamic patterns [128].

Interacting Gaussian Processes

To control a mobile robot to navigate through dense human crowds, [129] proposed the interacting GPs model, a prediction probability density function that captures interactions between a robot and humans in scenarios of cooperative collision avoidance. The robot is modeled as one of the humans, and a joint distribution describing their interaction is captured by the concatenation of independent GPs and an interaction potential, defined by the pairwise distance between any two decision makers (human–robot, human–human). The path-planning problem is then treated as an inference problem, with paths computed from the maximum likelihood estimate of the interacting GP, where the GP models a distribution over functions mapping the robot location to the corresponding velocity it should adopt. In [130], a computationally efficient approach to detecting change points in the GP framework is provided and has been demonstrated to work on the challenging problem of decision making in the presence of multiple models [131].

Deep Gaussian Processes

As shown in the previous section, a GP model is equivalent to an NN whenever the implicit features $h(z)$ of the kernel correspond to the (infinitely many) hidden units of a *one-layer* multilayer perceptron (MLP). More generally, *multilayer* NNs implement a composition of vector-valued, nonlinear functions. Such networks are called *deep* whenever they are composed of a large number of hidden layers; this results in powerful models such as convolution NNs and recurrent NNs. However, these models have parametric forms, which result in issues such as those discussed in "Limitations of Parametric Models—A Flight Control Example." Given the observation that a single GP models a one-layer NN, it is possible to construct a composition of GPs, where the outputs of one GP are used as inputs to a subsequent GP, to obtain *nonparametric* models that are equivalent to general multilayer NNs. Such composition of GPs can be interpreted as placing a prior over the values of the (possibly infinitely many) hidden units of an MLP. Concretely, sequencing GPs is equivalent to placing a prior over the composition of vector-valued functions, specifically, those encoded by the connections between inputs and outputs of the layers of the MLP. The outputs of a given layer of the MLP are drawn independently from a GP prior and used as inputs to a subsequent GP. DNNs have achieved significant success both in deep learning [132] and RL [98] tasks. Since NNs have been widely used (although with criticism) in the control community, it is foreseeable that the deep GP model may also play significant roles in solving control problems. Even though it has been noted [133] that single-GP models have limited generalization ability when they use predefined kernels,

TABLE 2 Various open-source Gaussian process (GP) implementations available for practitioners.

Package	Implementation	Authors	Description
GPML [137]	Matlab	C. Rasmussen and H. Nickisch	Implementation of the main algorithms presented in [26]
GPdyn [138]	Matlab	M. Stepančič and J. Kocijan	Toolbox for dynamic system identification with GP models
PILCO and GP TRSS [139]	Matlab	M. Densenroth	Policy search and filtering methods with GPs
GP IRL [140]	Matlab	S. Levine, Z. Popovic, and V. Koltun	Toolbox for Inverse RL with GPs
GPMRAC [141]	Matlab	G. Chowdhary	Budgeted online GP regression. This is the core GP engine used for the adaptive control approach in the section "Application I: Gaussian Processes for Model Reference Adaptive Control"
GPQ [113]	Matlab	G. Chowdhary and M. Liu	An implementation of GP Q-Learning (see the section "Application III: Reinforcement Learning—Gaussian Process Q-Learning)
GPy [142]	Python	N. Lawrence	Standard and advanced GP algorithms, including multiple output GPs, variational inference, and Markov chain Monte Carlo
GP in scikit-learn [143]	Python	—	Standard GP algorithms
SOGP [144]	C++	D. Grollman	Sparse online GP
Tpros [145]	C	D. MacKay and M. Gibbs	GP regression

such as square-exponential kernel (4), it is possible to exploit the idea of deep GPs to enhance the generalization capabilities of GPs. Specifically, the composition of GPs can be interpreted as a deep model that *constructs* kernels (based on data) by composing multiple layers of feature mappings. Kernel construction is related to the idea of kernel hyperparameter estimation, discussed in the section "Introduction to Gaussian Processes," where kernel parameters were adjusted to allow a GP to more closely capture the patterns in a given data set. When constructing a kernel, however, the analytic form of the kernel itself can be adjusted based on the data. In this sense, kernel construction (or kernel learning) is equivalent to representation learning [134]. See [133] and [135] for more details.

AVAILABLE RESOURCES FOR GAUSSIAN PROCESSES PRACTITIONERS

Many resources exist that can help practitioners use GPs to solve particular problems, including textbooks and online software. Popular implementations are summarized in Table 2. Other authors have also compiled more comprehensive lists of software resources for GP practitioners; a good complementary list can be found in [136].

CONCLUSION

This tutorial introduced GPs and GPR from the perspective of Bayesian nonparametric inference for use in control and RL problems. It emphasized the key features of GPRs, including 1) how they allow for automatic, data-driven feature selection, thereby not requiring practitioners to predefine feature numbers and locations, 2) the way in which GPs offer uncertainty measures of predictions, 3) how Bayesian inference allows for natural optimization and selection of GP hyperparameters, and 4) how GPs offer a principled way for performing budgeted online inference. The tutorial provided concrete examples demonstrating GPR for adaptive control and off-policy RL, and it also discussed the application of GPs to model value functions in optimal control and planning problems to model reward functions in inverse optimal control problems. Moreover, it showed how GPs can be used as the basis of a probabilistic framework capable of modeling system dynamics and measurement functions, with use in MPC and state estimation and filtering problems. In addition, the tutorial elaborated the connections between GPR and other widely used regression techniques, so that readers and practitioners are able to understand the unique features of GPs and their generality. Finally, the tutorial discussed a few limitations of the basic GPR approach and provided a brief overview of several advanced GP models that overcome such limitations. The discussions presented here are by no means exhaustive; our intention is merely to highlight the key features of GPs, through illustrative examples, to help interested practitioners more efficiently explore the relevant literature.

ACKNOWLEDGMENT

The research discussed here was supported in part by ONR MURI Grant N000141110688.

AUTHOR INFORMATION

Miao Liu (miao.liu1@ibm.com) is a research staff member in the AI Science Department at IBM T.J. Watson Research Center, Yorktown Heights, New York. Prior to joining IBM in 2016, he was a postdoctoral associate in the Laboratory of Information and Decision System at the Massachusetts Institute of Technology, where he worked on scalable Bayesian nonparametric methods for solving multiagent learning and planning problems. He received the B.S. and M.S. degrees in electronics and information engineering from Huazhong University of Science and Technology, Wuhan, China in 2005 and 2007, respectively, and the Ph.D. degree in electrical and computer engineering from Duke University, North Carolina, in 2014. He was a coauthor of the best student paper at IROS2017 and was nominated for the best multirobot paper in ICRA2017. His research interests include statistical machine learning, AI, and robotics. He can be contacted at IBM T.J. Watson Research Center, 1101 Kitchawan Rd. Room 34-245, P.O. Box 218, Yorktown Heights, NY 10598 USA.

Girish Chowdhary is an assistant professor at the University of Illinois at Urbana-Champaign (UIUC) and is affiliated with the Departments of Electrical and Computer Engineering and Agricultural and Biological Engineering and the UIUC Coordinated Science Laboratory. He is the director of the Distributed Autonomous Systems laboratory at UIUC. He received the Ph.D. degree in aerospace engineering in 2010 from the Georgia Institute of Technology and his undergraduate degrees from Royal Melbourne Institute of Technology, Australia. He was a postdoctoral associate at the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology from 2011 to 2013. He was an assistant professor at Oklahoma State University in the Department of Mechanical and Aerospace Engineering from 2013 to 2016. He also worked with the German Aerospace Center's Institute of Flight Systems from 2003 to 2006. His ongoing research interest is in theoretical insights and practical algorithms for adaptive autonomy.

Bruno Castro da Silva is an associate professor at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), Brazil. Previously, he was a postdoctoral associate at the Aerospace Controls Laboratory, Massachusetts Institute of Technology. He received the B.S. degree (cum laude) in computer science in 2004 and the M.Sc. degree in 2007, both from UFRGS, and the Ph.D. degree in computer science from the University of Massachusetts in 2014. He has been a visiting researcher at the Laboratory of Computational Embodied Neuroscience, Rome, Italy, developing novel control algorithms for the iCub robot. His research interests lie in the intersection of machine learning, reinforcement learning, optimal

control theory, and robotics and include the construction of reusable motor skills, active learning, efficient exploration of large state-spaces, and Bayesian optimization applied to control.

Shih-Yuan Liu is a senior research scientist at nuTonomoy, Inc. Previously, he was a postdoctoral associate at the Laboratory for Information and Decision Systems and Aerospace Controls Laboratory at the Massachusetts Institute of Technology. He received the Ph.D. degree in mechanical engineering in controls from University of California, Berkeley, in 2014. His research interests include control, path planning, coordination, and teleoperation of autonomous ground and aerial vehicles in dynamic environments.

Jonathan P. How is the Richard C. Maclaurin Professor of Aeronautics and Astronautics at the Massachusetts Institute of Technology (MIT). He received the B.A.Sc. degree from the University of Toronto in 1987 and the S.M. and Ph.D. degrees in aeronautics and astronautics from MIT in 1990 and 1993, respectively. He then studied for two years at MIT as a postdoctoral associate for the Middeck Active Control Experiment that flew onboard the Space Shuttle Endeavour in March 1995. Prior to joining MIT in 2000, he was an assistant professor in the Department of Aeronautics and Astronautics at Stanford University. He is the editor-in-chief of *IEEE Control Systems Magazine* and an associate editor for *AIAA Journal of Aerospace Information Systems* and *IEEE Transactions on Neural Networks and Learning Systems*. He received the 2002 Institute of Navigation Burka Award; a Boeing Special Invention award in 2008; the IFAC Automatica Award for best applications paper in 2011; the AeroLion Technologies Outstanding Paper Award for the Journal of Unmanned Systems in 2015; the IEEE Control Systems Society Video Clip Contest Award in 2015; and the AIAA Best Paper in Conference Awards in 2011, 2012, and 2013. He was a coauthor of the paper that won the Best Student Paper Award at IROS 2017. He is a Fellow of the IEEE and the AIAA.

REFERENCES

[1] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela, "A Bayesian nonparametric approach to adaptive control using Gaussian processes," in *Proc. IEEE 52nd Annu. Conf. Decision and Control,* 2013, pp. 874–879.

[2] X. Yang and J. M. Maciejowski, "Fault tolerant control using Gaussian processes and model predictive control," *Int. J. Appl. Math. Comput. Sci.*, vol. 25, no. 1, pp. 133–148, 2015.

[3] J. Joseph, F. Doshi-Velez, A. S. Huang, and N. Roy, "A Bayesian nonparametric approach to modeling motion patterns," *Auton. Robots*, vol. 31, no. 4, pp. 383–400, 2011.

[4] H. Wei, W. Lu, P. Zhu, S. Ferrari, R. H. Klein, S. Omidshafiei, and J. P. How, "Camera control for learning nonlinear target dynamics via Bayesian nonparametric Dirichlet-process Gaussian-process (DP-GP) models," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2014, pp. 95–102.

[5] D. Nguyen-Tuong and J. Peters. "Using model knowledge for learning inverse dynamics," in *Proc. IEEE Int. Conf. Robotics and Automation,* 2010, pp. 2677–2682.

[6] J. Ko and D. Fox, "GP-Bayes filters: Bayesian filtering using Gaussian process prediction and observation models," *Auton. Robots*, vol. 27, no. 1, pp. 75–90, 2009.

[7] J. Ko, D.J. Klein, D. Fox, and D. Haehnel, "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," in *Proc. IEEE Int. Conf. Robotics and Automation,* 2007, pp. 742–747.

[8] J. Ko and D. Fox, "Learning GP-Bayesfilters via Gaussian process latent variable models," *Auton. Robots*, vol. 30, no. 1, pp. 3–23, 2011.

[9] N. Xu, K. H. Low, J. Chen, K. K. Lim, and E. B. Ozgul, "GP-Localize: Persistent mobile robot localization using online sparse Gaussian process observation model," in *Proc. 30th Conf. Association Advancement of Artificial Intelligence*, 2014, pp. 2585–2592.

[10] M. P. 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, 2015.

[11] D. Lizotte, T. Wang, M. Bowling, and D. Schuurmans, "Automatic gait optimization with Gaussian process regression," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2007, vol. 7, pp. 944–949.

[12] M. Tesch, J. Schneider, and H. Choset, "Using response surfaces and expected improvement to optimize snake robot gait parameters," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2011, pp. 1069–1074.

[13] G. Tao, *Adaptive Control Design and Analysis*. New York: Wiley, 2003.

[14] K. S. Narendra and A. M. Annaswamy, *Stable Adaptive Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[15] P. A. Ioannou and J. Sun, *Robust Adaptive Control*. Upper Saddle River, NJ: Prentice-Hall, 1996.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2017.

[17] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, vol. 39. Boca Raton, FL: CRC, 2010.

[18] D. Vrabie, K. G. Vamvoudakis, and F. L. Lewis, *Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles*, vol. 2. United Kingdom: IET, 2013.

[19] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *IEEE Control Syst.*, vol. 32, no. 6, pp. 76–105, 2012.

[20] S. Bhasin, R. Kamalapurkar, M. Johnson, K. G. Vamvoudakis, F. L. Lewis, and W. E. Dixon, "A novel actor–critic–identifier architecture for approximate optimal control of uncertain nonlinear systems," *Automatica*, vol. 49, no. 1, pp. 82–92, 2013.

[21] K. G. Vamvoudakis, H. Modares, B. Kiumarsi, and F. L. Lewis, "Game theory-based control system algorithms with real-time reinforcement learning: How to solve multiplayer games online," *IEEE Control Syst.*, vol. 37, no. 1, pp. 33–52, 2017.

[22] D. Liu, Q. Wei, D. Wang, X. Yang, and H. Li, *Adaptive Dynamic Programming with Applications in Optimal Control*. New York: Springer, 2017.

[23] Y Zhu and D Zhao, "Comprehensive comparison of online ADP algorithms for continuous-time optimal control," *Artif. Intell. Rev.*, vol. 49, no. 4, pp. 531–547, Apr. 2018.

[24] R. M. Sanner and J. E. Slotine, "Gaussian networks for direct adaptive control," *IEEE Trans. Neural Netw.*, vol. 3, no. 6, pp. 837–863, 1992.

[25] Y. H. Kim and F. L. Lewis, *High-Level Feedback Control with Neural Networks*. Singapore: World Scientific, 1998.

[26] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2005.

[27] N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*, vol. 2. Cambridge, MA: MIT Press, 1949.

[28] A. Kolmogoroff, "Interpolation und extrapolation von stationaren zufalligen folgen," *Izvestiya Rossiiskoi Akademii Nauk. Seriya Matematicheskaya*, vol. 5, no. 1, pp. 3–14, 1941.

[29] G. Matheron, "The intrinsic random functions and their applications," *Adv. Appl. Probab.*, vol. 5, no. 3, pp. 439–468, Dec. 1973.

[30] A. G. Journel and C. J. Huijbregts, *Mining Geostatistics*. New York: Academic, 1978.

[31] N. Aronszajn, "Theory of reproducing kernels," *Trans. Amer. Math. Soc.*, vol. 68, no. 3, pp. 337–404, May 1950.

[32] Y. H. Kim and F. L. Lewis, *High-Level Feedback Control with Neural Networks, Volume 21 of Robotics and Intelligent Systems*. Singapore: World Scientific, 1998.

[33] H. A. Kingravi, G. Chowdhary, P. A. Vela, and E. N. Johnson, "Reproducing kernel Hilbert space approach for the online update of radial bases in neuro-adaptive control," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 7, pp. 1130–1141, July 2012.

[34] K. S. Narendra, "Neural networks for control theory and practice," *Proc. IEEE*, vol. 84, no. 10, pp. 1385–1406, Oct. 1996.

[35] K. Y. Volyanskyy, W. M. Haddad, and A. J. Calise, "A new neuroadaptive control architecture for nonlinear uncertain dynamical systems:

Beyond σ and e-modifications," *IEEE Trans. Neural Netw.*, vol. 20, no. 11, pp. 1707–1723, Nov. 2009.
[36] 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.
[37] K. J. Aström and B. Wittenmark, *Adaptive Control*, 2nd ed. Reading, MA: Addison-Wesley, 1995.
[38] S. Boyd and S. Sastry, "Necessary and sufficient conditions for parameter convergence in adaptive control," *Automatica*, vol. 22, no. 6, pp. 629–639, 1986.
[39] K. S. Narendra and A. M. Annaswamy, "Robust adaptive control in the presence of bounded disturbances," *IEEE Trans. Autom. Control*, vol. AC-31, no. 4, pp. 306–315, 1986.
[40] N. Lawrence, "Probabilistic non-linear principal component analysis with Gaussian process latent variable models," *J. Mach. Learn. Res.*, vol. 6, pp. 1783–1816, Nov. 2005.
[41] R. Murray-Smith and D. Sbarbaro, "Nonlinear adaptive control using non-parametric Gaussian process prior models," in *Proc. 15th Triennial World Congr. Int. Federation Automatic Control*, 2002, pp. 325–330.
[42] R. Murray-Smith, D. Sbarbaro, C. E. Rasmussen, and A. Girard, "Adaptive, cautious, predictive control with Gaussian process priors," in *Proc. 13th Int. Federation Automatic Control Symp. System Identification*, 2003, pp. 1195–1200.
[43] J. Ko, D. J. Klein, D. Fox, and D. Hähnel, "GP-UKF: Unscented Kalman filters with Gaussian process prediction and observation models," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2007, pp. 1901–1907.
[44] M. P. Deisenroth, C. E. Rasmussen, and J. Peters, "Gaussian process dynamic programming," *Neurocomputing*, vol. 72, no. 7, pp. 1508–1524, 2009.
[45] M. Deisenroth, D. Fox, and C. 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.
[46] D. Nguyen-Tuong and J. Peters, "Local Gaussian process regression for real-time model-based robot control," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2008, pp. 380–385.
[47] D. J. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2003.
[48] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press, 2012.
[49] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, 2002.
[50] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian Data Analysis*, vol. 2. New York: Taylor & Francis, 2014.
[51] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics*). New York: Springer-Verlag, 2006.
[52] L. Csató and M. Opper, "Sparse on-line Gaussian processes," *Neural Comput.*, vol. 14, no. 3, pp. 641–668, 2002.
[53] J. Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *J. Mach. Learn. Res.*, vol. 6, pp. 1939–1959, Nov. 2005.
[54] S. Haykin, *Neural Networks a Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1998.
[55] I. Goodfellow, Y. Bengio, and A. Courville. (2016). *Deep Learning*. Cambridge, MA: MIT Press. [Online]. Available: http://www.deeplearningbook.org
[56] E. Johnson and S. Kannan, "Adaptive trajectory control for autonomous helicopters," *J. Guid. Control Dyn.*, vol. 28, no. 3, pp. 524–538, May 2005.
[57] A. Calise, N. Hovakimyan, and M. Idan, "Adaptive output feedback control of nonlinear systems using neural networks," *Automatica*, vol. 37, no. 8, pp. 1201–1211, 2001.
[58] T. Zhang, S. S. Ge, and C. C. Hang, "Direct adaptive control of nonaffine nonlinear systems using multilayer neural networks," in *Proc. American Control Conf.*, June 1998, vol. 1, pp. 515–519.
[59] K. Nakawan, "Improved methods in neural network based adaptive output feedback control, with applications to flight control," Ph.D. dissertation, Georgia Inst. Technol., Atlanta, GA, 2003.
[60] G. Chowdhary, "Concurrent learning for convergence in adaptive control without persistency of excitation," Ph.D. dissertation, Georgia Inst. Technol., Atlanta, GA, Dec. 2010.
[61] G. Chowdhary, M. Mühlegg, and E. N. Johnson, "Exponential parameter and tracking error convergence guarantees for adaptive controllers without persistency of excitation," *Int. J. Control*, vol. 87, no. 8, pp. 1583–1604, 2014.
[62] N. Hovakimyan and C. Cao, *1 Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation*. Philadelphia, PA: SIAM, 2010.
[63] N. Nguyen, "Asymptotic linearity of optimal control modification adaptive law with analytical stability margins," in *Proc. Infotech@ American Institute Aeronautics and Astronautics Conf.*, Atlanta, GA, 2010. pp. 1–4.
[64] E. Lavretsky, "Combined/composite model reference adaptive control," *IEEE Trans. Autom. Control*, vol. 54, no. 11, pp. 2692–2707, Nov. 2009.
[65] R. M. Sanner and J.-J. E. Slotine, "Gaussian networks for direct adaptive control," *IEEE Trans. Neural Netw.*, vol. 3, no. 6, pp. 837–863, Nov. 1992.
[66] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Comput.*, vol. 3, no. 2, pp. 246–257, June 1991.
[67] J.-B. Pomet and L. Praly, "Adaptive nonlinear regulation: Estimation from the Lyapunov equation," *IEEE Trans. Autom. Control*, vol. 37, no. 6, pp. 729–740, June 1992.
[68] M. A. Alvarez, L. Rosasco, and N. D. Lawrence, "Kernels for vector-valued functions: A review," arXiv preprint, arXiv:1106.6251, 2011.
[69] P. Boyle and M. Frean, "Dependent Gaussian processes," in *Proc. Advances Neural Information Processing Systems*, 2004, pp. 217–224.
[70] A. Gelb, *Applied Optimal Estimation*. Cambridge, MA: MIT Press, 1974.
[71] G. Chowdhary and E. N. Johnson, "Theory and flight test validation of a concurrent learning adaptive controller," *J. Guid. Control Dyn.*, vol. 34, no. 2, pp. 592–607, Mar. 2011.
[72] R. C. Grande, G. Chowdhary, and J. P. How, "Experimental validation of Bayesian nonparametric adaptive control using Gaussian processes," *J. Aerosp. Inform. Syst.*, vol. 11, no. 9, pp. 565–578, 2014.
[73] R. C. Grande, G. Chowdhary, and J. P. How, "Nonparametric adaptive control using Gaussian processes with online hyperparameter estimation," in *Proc. IEEE 52nd Annu. Conf. Decision and Control*, 2013, pp. 861–867.
[74] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela, "A Bayesian nonparametric approach to adaptive control using Gaussian processes," in *Proc. IEEE 52nd Annu. Conf. Decision and Control*, pp. 874–879, 2013.
[75] R. C. Grande, G. Chowdhary, and J. P. How, "Nonparametric adaptive control using Gaussian processes with online hyperparameter estimation," in *Proc. IEEE 52nd Annu. Conf. Decision and Control*, 2013, pp. 861–867.
[76] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Syst.*, vol. 28, no. 2, pp. 51–64, 2008.
[77] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Syst.*, vol. 28, no. 2, pp. 51–64, Apr. 2008.
[78] M. Cutler, "Design and control of an autonomous variable-pitch quadrotor helicopter," M.S. thesis, Dept. Aeronautics and Astronautics, Massachusetts Inst. Technol., Aug. 2012.
[79] M. Cutler and J. P. How, "Actuator constrained trajectory generation and control for variable-pitch quadrotors," in *Proc. American Institute Aeronautics and Astronautics Guidance Navigation and Control Conf.*, Minneapolis, MN, Aug. 2012.
[80] G. Chowdhary, T. Wu, M. Cutler, and J. P. How, "Rapid transfer of controllers between UAVS using learning-based adaptive control," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2013, pp. 5409–5416.
[81] G. Chowdhary and E. N. Johnson, "Concurrent learning for convergence in adaptive control without persistency of excitation," in *Proc. 49th IEEE Conf. Decision and Control*, 2010, pp. 3674–3679.
[82] 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, June 2010.
[83] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," *Adv. Neural Inform. Process. Syst.*, vol. 18, pp. 1257–1264, Dec. 2005.
[84] F. Nardi, "Neural network based adaptive algorithms for nonlinear control," Ph.D. dissertation, Georgia Inst. Technol., School Aerosp. Eng., Atlanta, GA, 2000.
[85] P. Shankar, "Self-organizing radial basis function networks for adaptive flight control and aircraft engine state estimation," Ph.D. dissertation, The Ohio State Univ., Columbus, OH, 2007.
[86] N. Sundararajan, P. Saratchandran, and L. Yan, *Fully Tuned Radial Basis Function Neural Networks for Flight Control*. New York: Springer, 2002.
[87] L. Ljung, "Analysis of recursive stochastic algorithms," *IEEE Trans. Autom. Control*, vol. 22, no. 4, pp. 551–575, Aug. 1977.
[88] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1. Belmont, MA: Athena Scientific, 1995.
[89] J. B. Rawlings, "Tutorial overview of model predictive control," *IEEE Control Syst. Mag.*, vol. 20, no. 3, pp. 38–52, 2000.
[90] M. Cutler, "Reinforcement learning for robots through efficient simulator sampling," Ph.D. dissertation, Dept. Aeronautics and Astronautics, Massachusetts Inst. Technol., Cambridge, MA, Aug. 2015.

[91] J. Kocijan, *Modelling and Control of Dynamic Systems Using Gaussian Process Models*. New York: Springer, 2015.
[92] M. Ghavamzadeh, S. Mannor, J. Pineau, and A. Tamar, *Bayesian Reinforcement Learning: A Survey*. Singapore: World Scientific, 2015.
[93] B. B. Doll, D. A. Simon, and N. D. Daw, "The ubiquity of model-based reinforcement learning," *Current Opinion Neurobiol.*, vol. 22, no. 6, pp. 1075–1081, 2012.
[94] C. J. Watkins, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, 1992.
[95] R. Sutton and A. Barto, *Reinforcement Learning, an Introduction*. Cambridge, MA: MIT Press, 1998.
[96] H. Modares, F. L. Lewis, and Z.-P. Jiang, "h_∞ tracking control of completely unknown continuous-time systems via off-policy reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 10, pp. 2550–2562, 2015.
[97] J. Škach, B. Kiumarsi, F. L. Lewis, and O. Straka, "Actor-critic off-policy learning for optimal control of multiple-model discrete-time systems," *IEEE Trans. Cybern.*, vol. 48, no. 1, pp. 29–40, Jan. 2018.
[98] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, and D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[99] L. C. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *Proc. Int. Conf. Machine Learning*, 1995, pp. 30–37.
[100] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal difference learning with function approximation," *IEEE Trans. Autom. Control*, vol. 42, no. 5, pp. 674–690, May 1997.
[101] F. S. Melo, S. P. Meyn, and M. I. Ribeiro. "An analysis of reinforcement learning with function approximation," in *Proc. Int. Conf. Machine Learning*, 2008, pp. 664–671.
[102] G. Chowdhary, M. Liu, R. Grande, T. Walsh, J. P. How, and L. Carin, "Off-policy reinforcement learning with Gaussian processes," *IEEE/CAA J. Autom. Sin.*, vol. 1, no. 3, pp. 227–238, 2014.
[103] R. Grande, T. Walsh, and J. How, "Sample efficient reinforcement learning with Gaussian processes," in *Proc. Int. Conf. Machine Learning*, 2014, pp. 1332–1340.
[104] R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora, "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *Proc. Int. Conf. Machine Learning*, 2009, pp. 993–1000.
[105] H. R. Maei, C. Szepesvïri, S. Bhatnagar, and R. S. Sutton, "Toward off-policy learning control with function approximation," in *Proc. Int. Conf. Machine Learning*, Haifa, Israel, 2010, pp. 719–726.
[106] J. Zico Kolter and A. Y. Ng, "Regularization and feature selection in least-squares temporal difference learning," in *Proc. 26th Annu. Int. Conf. Machine Learning*, 2009, pp. 521–528.
[107] B. Liu, S. Mahadevan, and J. Liu, "Regularized off-policy TD-learning," in *Proc. Neural Information and Processing Systems*, Lake Tahoe, NV, Dec. 2012, pp. 836–844.
[108] Y. Engel, S. Mannor, and R. Meir, "Bayes meets Bellman: The Gaussian process approach to temporal difference learning," in *Proc. 20th Int. Conf. Machine Learning*, 2003, pp. 154–161.
[109] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *Proc. Int. Conf. Machine Learning*, 2005,pp. 201–208.
[110] Y. Engel, P. Szabo, and D. Volkinshtein. "Learning to control an octopus arm with Gaussian process temporal difference methods," in *Proc. Advances Neural Information Processing Systems*, 2005, p. 18.
[111] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, Apr. 2005.
[112] A. L. Strehl and M. L. Littman. "A theoretical analysis of model-based interval estimation," in *Proc. Int. Conf. Machine Learning*, 2005, pp. 856–863.
[113] PPQ. (2013). GP Q-learning. [Online]. Available: https://github.com/GPCSM/GPQ
[114] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, Dec. 2003.
[115] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proc. Int. Conf. Machine Learning*, 2000, pp. 663–670.
[116] R. E. Kalman, "When is a linear control system optimal," *J. Basic Eng.*, vol. 86, no. 1, pp. 51–60, 1964.
[117] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Philadelphia, PA: SIAM, 1994.
[118] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. Association Advancement Artificial Intelligence*, Chicago, IL, 2008, vol. 8, pp. 433–1438.
[119] S. Levine, Z. Popovic, and V. Koltun,"Nonlinear inverse reinforcement learning with Gaussian processes," in *Proc. Advances Neural Information Processing Systems*, 2011, pp. 19–27.
[120] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," in *Proc. ACM 29th Int. Conf. Machine Learning*, New York, 2012, pp. 41–48.
[121] Q. Qiao and P. A. Beling, "Inverse reinforcement learning with Gaussian process," in *Proc. IEEE American Control Conf.*, 2011, pp. 113–118.
[122] B. Michini, T. J. Walsh, A.-A. Agha-Mohammadi, and J. P. How, "Bayesian nonparametric reward learning from demonstration," *IEEE Trans. Robot.*, vol. 31, no. 2, pp. 369–386, 2015.
[123] S. Ferguson, B. Luders, R. C. Grande, and J. P. How, "Real-time predictive modeling and robust avoidance of pedestrians with uncertain, changing intentions," in *Proc. Workshop Algorithmic Foundations Robotics*, Istanbul, Turkey, Aug. 2014, pp. 161–177.
[124] S.-Y Liu. (2015). Dependent GPs. [Online]. Available: https://github.com/GPCSM/DGP
[125] K. Kim, D. Lee, and I. Essa, "Gaussian process regression flow for analysis of motion trajectories," in *Proc. IEEE Int. Conf. Computer Vision*, 2011, pp. 1164–1171.
[126] C. E. Rasmussen and Z. Ghahramani, "Infinite mixtures of Gaussian process experts," *Adv. Neural Inform. Process. Syst.*, vol. 2, pp. 881–888, Dec. 2002.
[127] C. D. McKinnon and A. P. Schoellig, "Learning multimodal models for robot dynamics online with a mixture of Gaussian process experts," in *Proc. IEEE Int. Conf. Robotics and Automation*, May 2017, pp. 322–328.
[128] H. Wei, W. Lu, P. Zhu, S. Ferrari, M. Liu, R. H. Klein, S. Omidshafiei, and J. P. How, "Information value in nonparametric Dirichlet-process Gaussian-process (DPGP) mixture models," *Automatica*, vol. 74, pp. 360–368, Dec. 2016.
[129] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation," *Int. J. Robot. Res.*, vol. 34, no. 3, pp. 335–356, 2015.
[130] R. C. Grande, T. J. Walsh, G. Chowdhary, S. Ferguson, and J. P. How, "Online regression for data with changepoints using Gaussian processes and reusable models," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 9, pp. 2115–2128, 2017.
[131] R. Allamaraju, H. Kingravi, A. Axelrod, G. Chowdhary, R. Grande, C. Crick, W. Sheng, and J. How, "Human aware path planning in urban environments with nonstationary MDPS," in *Proc. IEEE Int. Conf. Robotics and Automation*, Hong Kong, China, 2014, pp. 1161–1167.
[132] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[133] D. K. Duvenaud, O. Rippel, R. P. Adams, and Z. Ghahramani, "Avoiding pathologies in very deep networks," *Artif. Intell. Stat.*, vol. 33, pp. 202–210, Apr. 2014.
[134] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, 2013.
[135] N. D. Lawrence. (2017). Deep probabilistic modeling with Gaussain processes. [Online]. Available: http://inverseprobability.com/talks/notes/deep-probabilistic-modelling-with-gaussian-processes.html
[136] GP Software (2006). A comprehensive list of software related to GPs. [Online]. Available: http://www.gaussianprocess.org/#related
[137] C. E. Rasmussen and C. Williams. (2006). GPML Matlab code. [Online]. Available: http://www.gaussianprocess.org/gpml/code/matlab/doc/index.html
[138] Martin Stepancic and Juš Kocijan (2016). GP-model-based system-identification toolbox for Matlab. [Online]. Available: https://github.com/Dynamic-Systems-and-GP/GPdyn
[139] M. Deisenroth. (2013). GP-based policy search and filtering. [Online]. Available: http://mloss.org/software/author/marc-deisenroth/
[140] S. Levine, Z. Popovic, and V. Koltun. (2011). GPIRL toolbox. [Online]. Available: http://graphics.stanford.edu/projects/gpirl/index.htm
[141] GP-MRAC. (2012). GP-MRAC toolbox. [Online]. Available: https://github.com/GPCSM/GPMRAC
[142] GPy. (2012). The GP framework in Python. [Online]. Available: https://github.com/SheffieldML/GPy
[143] The GP in scikit-learn. [Online]. Available: http://scikit-learn.org/stable/modules/gaussian_process.html
[144] D. Grollman. (2015). Sogp: GP implementation in C++. [Online]. Available: https://github.com/cbecker/SOGP
[145] M. Gibbs. (2009). Tpros: GP implementation in C. [Online]. Available: http://wol.ra.phy.cam.ac.uk/mng10/GP/GP.html