Auton Robot (2011) 30: 3–23
DOI 10.1007/s10514-010-9213-0

Learning GP-BayesFilters via Gaussian process latent variable models

Jonathan Ko · Dieter Fox

Received: 31 December 2009 / Accepted: 1 October 2010 / Published online: 26 October 2010

Abstract GP-BayesFilters are a general framework for integrating Gaussian process prediction and observation models into Bayesian filtering techniques, including particle filters and extended and unscented Kalman filters. GP-BayesFilters have been shown to be extremely well suited for systems for which accurate parametric models are difficult to obtain. GP-BayesFilters learn non-parametric models from training data containing sequences of control inputs, observations, and ground truth states. The need for ground truth states limits the applicability of GP-BayesFilters to systems for which the ground truth can be estimated without significant overhead. In this paper we introduce GPBF-LEARN, a framework for training GP-BayesFilters without ground truth states. Our approach extends Gaussian Process Latent Variable Models to the setting of dynamical robotics systems. We show how weak labels for the ground truth states can be incorporated into the GPBF-LEARN framework. The approach is evaluated using a difficult tracking task, namely tracking a slotcar based on inertial measurement unit (IMU) observations only. We also show some special features enabled by this framework, including time alignment, and control replay for both the slotcar, and a robotic arm.

Keywords Gaussian process · System identification · Bayesian filtering · Time alignment · System control · Machine learning

J. Ko (✉) · D. Fox
Department of Computer Science & Engineering, University of Washington, Seattle, WA, USA
e-mail: jonko@cs.washington.edu

D. Fox
Intel Labs Seattle, Intel Corp., Seattle, WA, USA

1 Introduction

Over the last years, Gaussian processes (GPs) have been applied with great success to robotics tasks such as reinforcement learning (Engel et al. 2006) and learning of prediction and observation models (Ferris et al. 2006; Ko et al. 2007; Plagemann et al. 2007). GPs learn probabilistic regression models from training data consisting of input-output examples (Rasmussen and Williams 2005). GPs combine extreme modeling flexibility with consistent uncertainty estimates, which makes them an ideal tool for learning of probabilistic estimation models in robotics. The fact that GP regression models provide Gaussian uncertainty estimates for their predictions allows them to be seamlessly incorporated into probabilistic filtering techniques, most easily into particle filters (Ferris et al. 2006; Plagemann et al. 2007).

GP-BayesFilters are a general framework for integrating Gaussian process prediction and observation models into Bayesian filtering techniques, including particle filters and extended and unscented Kalman filters (Ko and Fox 2008; Ko et al. 2007). More recently, the GP-BayesFilter framework has been extended to also include assumed density filters (ADF) (Deisenroth et al. 2009). GP-BayesFilters learn GP filter models from training data containing sequences of control inputs, observations, and ground truth states. In the context of tracking a micro-blimp, GP-BayesFilters have been shown to provide excellent performance, significantly outperforming their parametric Bayes filter counterparts. Furthermore, GP-BayesFilters can be combined with parametric models to improve data efficiency and thereby reduce computational complexity (Ko and Fox 2008). However, the need for ground truth training data requires substantial labeling effort or special equipment such as a motion capture system in order to determine the true state of the system during training (Ko et al. 2007). This requirement limits the

Fig. 1 (Color online) (*Left*) The slotcar track used during the experiments. An overhead camera supplies ground truth locations of the car. (*Right*) The test vehicle moves along a slot in the track, velocity control is provided remotely by a desktop PC. The state of the vehicle is estimated based on an on-board IMU (indicated by *the red outline*)

applicability of GP-BayesFilters to systems for which such ground truth states are readily available.

The need for ground truth states in GP-BayesFilter training stems from the fact that standard GPs only model noise in the output data, input training points are assumed to be noise-free (Rasmussen and Williams 2005). To overcome this limitation, Lawrence recently introduced Gaussian Process Latent Variable Models (GPLVM) for probabilistic, non-linear principal component analysis (Lawrence 2005). In contrast to the standard GP training setup, GPLVMs only require output training examples; they determine the corresponding inputs via optimization. Just like other dimensionality reduction techniques such as principal component analysis (PCA), GPLVMs learn an embedding of the output examples in a low-dimensional latent (input) space. In contrast to PCA, however, the mapping from latent space to output space is not a linear function but a Gaussian process. While GPLVMs were originally developed in the context of visualization of high-dimensional data, recent extensions enabled their application to dynamic systems (Ferris et al. 2007; Lawrence and Moore 2007; Urtasun et al. 2006; Wang et al. 2008).

In this paper we introduce GPBF-LEARN, a framework for learning GP-BayesFilters from partially or fully unlabeled training data. The inputs to GPBF-LEARN are temporal sequences of observations and control inputs along with partial information about the underlying state of the system. GPBF-LEARN proceeds by first determining a state sequence that best matches the control inputs, observations, and partial labels. These states are then used along with the control and observations to learn a GP-BayesFilter, just as in Ko and Fox (2008). Partial information ranges from noisy ground truth states, to sparse labels in which only a subset of the states are labeled, to completely label-free data. To determine the optimal state sequence, GPBF-LEARN extends recent advances in GPLVMs to incorporate robot control information and probabilistic priors over the hidden states.

Under our framework, alignment of multiple time series and filtering from completely unlabeled data is possible. Furthermore, we describe a method for control replay using GPBF-LEARN from multiple user demonstrations. We demonstrate the capabilities of GPBF-LEARN using the autonomous slotcar testbed shown in Fig. 1. The car moves along a slot on a race track while being controlled remotely. Position estimation is performed based on an inertial measurement unit (IMU) placed on the car. Note that tracking solely based on the IMU is difficult, since the IMU provides only noisy acceleration and turn information. Using this testbed, we demonstrate that GPBF-LEARN outperforms alternative approaches to learning GP-BayesFilters.

This paper is an extension of a previous work (Ko and Fox 2009). Significant additions include a description of GPBF-LEARN for time alignment of time series data, and control replay of expert demonstrations. The paper is also augmented with substantial additional experimental results.

This paper is organized as follows: after discussing related work, we provide background on Gaussian process regression, Gaussian process latent variable models, and GP-BayesFilters. Then, in Sect. 4, we introduce the GPBF-LEARN framework. Experimental results are given in Sect. 5, followed by a discussion.

2 Related work

Lawrence introduced Gaussian Process Latent Variable Models (GPLVMs) for visualization of high-dimensional data (Lawrence 2005). Original GPLVMs impose no smoothness constraints on the latent space. They are thus not able to take advantage of the temporal nature of dynamical systems. One way to overcome this limitation is the introduction of so-called back-constraints (Lawrence and Quiñonero Candela 2006), which have been applied successfully in the context of WiFi-SLAM, where the goal is to learn an observation model for wireless signal strength data without relying on ground truth location data (Ferris et al. 2007).

Wang and colleagues (Wang et al. 2008) introduced Gaussian Process Dynamic Models (GPDM), which are an extension of GPLVMs specifically aimed at modeling dynamical systems. GPDMs have been applied successfully to computer animation (Wang et al. 2008) and visual tracking

(Urtasun et al. 2006) problems. However, these models do not aim at tracking the hidden state of a physical system, but rather at generating good observation sequences for animation. They are thus not able to incorporate control input or information about the desired structure of the latent space. Furthermore, the tracking application introduced by Urtasun and colleagues (Urtasun et al. 2006) is not designed for real-time or near real-time performance, nor does it provide uncertainty estimates as GP-BayesFilters. Other alternatives for non-linear embedding in the context of dynamical systems are hierarchical GPLVMs (Lawrence and Moore 2007) and action respecting embeddings (ARE) (Bowling et al. 2005). None of these techniques are able to incorporate control information or impose prior knowledge on the structure of the latent space. We consider both capabilities to be extremely important for robotics applications.

The system identification community has developed various subspace identification techniques (Ljung 1987; Van Overschee and De Moor 1996). The goal of these techniques is the same as that of GPBF-LEARN when applied to label-free data, namely to learn a model for a dynamical system from sequences of control inputs and observations. The model underlying N4SID (Van Overschee and De Moor 1996) is linear, and the parameters learned can be used to instantiate a linear Kalman filter. Due to its flexibility and robustness, N4SID is extremely popular. It has been applied successfully for human motion animation (Hsu et al. 2005). In our experiments, we demonstrate that GPBF-LEARN provides superior performance due to its ability to model non-linear systems. We show however that N4SID provides good initialization for GPBF-LEARN.

Other models for learning dynamical systems do exist. Predictive state representations (PSRs) are models in which the "state" of the system is grounded in statistics over observations (Littman et al. 2001). They do not explicitly keep track of a hidden or latent state. More recently, PSRs have been used for planning in a constrained real-world problem, where a robot learns to navigate an environment using a PSR model (Boots et al. 2009). Other non-linear system identification techniques are also well researched. These non-linear techniques use different functional bases including wavelets and neural networks. A thorough overview can be found in Sjöberg et al. (1995).

Time alignment of time series data can be performed using GPBF-LEARN. This is an interesting feature of the GPBF-LEARN algorithm, as it is a fundamental problem in the speech recognition community, where it is known as dynamic time warping (Rabiner et al. 1978). Time alignment is also an important algorithm for human motion analysis (Hsu et al. 2007; Zhou and De la Torre 2009). The main difference between these alignment algorithms and GPBF-LEARN alignment is that they have no notion of dynamics models and control inputs as required for robotic systems. Our technique is most closely related to time alignment in other robotics applications like (Coates et al. 2008) and (Schmill et al. 1999).

A fundamental problem in robotics is to make a robot perform a desired action. Two common ways of doing this are by explicit programming, or by using reinforcement learning techniques. Recently, imitation learning has become very prominent, where robots learn from the behavior demonstrated by a teacher. We show the use of the GPBF-LEARN framework for control replay of human demonstrations. This is similar in spirit to other robot imitation learning systems, such as Grimes and Rao (2008) where a humanoid robot learns from human demonstrations, Ekvall and Kragic (2004) where robot grasps are learned from human trials, or Abbeel et al. (2008) where a robotic car learns from human drivers.

3 Preliminaries

This section provides background on Gaussian processes for regression, their extension to latent variable models (GPLVMs), and GP-BayesFilters, which use GP regression to learn observation and prediction models for Bayesian filtering.

3.1 Gaussian process regression

Gaussian processes (GPs) are non-parametric techniques for learning regression functions from sample data (Rasmussen and Williams 2005). Assume we have n d-dimensional input vectors:

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]. \tag{1}$$

A GP defines a zero-mean, Gaussian prior distribution over the outputs $\mathbf{y} = [y_1, y_2, \ldots, y_n]$ at these values:[1]

$$p(\mathbf{y} \mid \mathbf{X}) = \mathcal{N}(\mathbf{y};\, 0,\, \mathbf{K}_y + \sigma_n^2 \mathbf{I}). \tag{2}$$

The covariance of this Gaussian distribution is defined via a kernel matrix, $\mathbf{K}_y$, and a diagonal matrix with elements σ_n^2 that represent zero-mean, white output noise. The elements of the $n \times n$ kernel matrix $\mathbf{K}_y$ are specified by a kernel function over the input values: $\mathbf{K}_y[i, j] = k(\mathbf{x}_i, \mathbf{x}_j)$. By interpreting the kernel function as a similarity measure, we see that if input points $\mathbf{x}_i$ and $\mathbf{x}_j$ are close in the kernel space, their output values y_i and y_j are highly correlated.

The specific choice of the kernel function k depends on the application, the most widely used being the squared exponential, or Gaussian, kernel:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}')W(\mathbf{x}-\mathbf{x}')^T}. \tag{3}$$

[1] For ease of exposition, we will only describe GPs for one-dimensional outputs, multi-dimensional outputs will be handled by assuming independence between the output dimensions.

We use this kernel function exclusively on all our experiments. The kernel function is parameterized by W and σ_f. The diagonal matrix W defines the length scales of the process, which reflect the relative smoothness of the process along the different input dimensions. Signal variance is denoted by σ_f^2.

Given training data $D = \langle \mathbf{X}, \mathbf{y} \rangle$ of n input-output pairs, a key task for a GP is to generate an output prediction at a test input $\mathbf{x}_*$. It can be shown that conditioning (2) on the training data and $\mathbf{x}_*$ results in a Gaussian predictive distribution over the corresponding output y_*

$$p(y_* \mid \mathbf{x}_*, D) = \mathcal{N}(y_*; \mathrm{GP}_\mu(\mathbf{x}_*, D), \mathrm{GP}_\Sigma(\mathbf{x}_*, D)) \tag{4}$$

with mean

$$\mathrm{GP}_\mu(\mathbf{x}_*, D) = \mathbf{k}_*^T [K + \sigma_n^2 I]^{-1} \mathbf{y} \tag{5}$$

and variance

$$\mathrm{GP}_\Sigma(\mathbf{x}_*, D) = \quad k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T [K + \sigma_n^2 I]^{-1} \mathbf{k}_*. \tag{6}$$

Here, $\mathbf{k}_*$ is a vector of kernel values between $\mathbf{x}_*$ and the training inputs $\mathbf{X}$: $\mathbf{k}_*[i] = k(\mathbf{x}_*, \mathbf{x}_i)$. Note that the prediction uncertainty, captured by the variance GP_Σ, depends on both the process noise and the correlation between the test input and the training inputs.

The hyperparameters $\boldsymbol{\theta}_y$ of the GP are given by the parameters of the kernel function and the output noise: $\boldsymbol{\theta}_y = \langle \sigma_n, W, \sigma_f \rangle$. They are typically determined by maximizing the log likelihood of the training outputs (Rasmussen and Williams 2005). Making the dependency on hyperparameters explicit, we get

$$\boldsymbol{\theta}_y^* = \underset{\boldsymbol{\theta}_y}{\operatorname{argmax}} \log \; p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}_y). \tag{7}$$

The GPs described thus far depend on the availability of fully labeled training data, that is, data containing ground truth input values $\mathbf{X}$ and possibly noisy output values $\mathbf{y}$.

Like most kernel methods, the use of Gaussian processes do have drawbacks in terms of learning and prediction efficiency. The training complexity is $O(n^3)$ for this basic formulation of GPs, and $O(n)$ and $O(n^2)$ for mean and variance predictions, respectively. Fortunately, much research has been directed at making GPs more efficient. A few recent papers show diversity of such approaches. In Snelson and Ghahramani (2006), Snelson and colleagues increase efficiency by learning using only representative "pseudo-inputs" from the full dataset. Complexity is reduced by learning multiple local GPs in Nguyen-Tuong et al. (2008). Finally, Rahimi and colleagues describes the use of random features for very large kernel machines that may be applicable to Gaussian processes and can speed up computation dramatically (Rahimi and Recht 2007).

3.2 Gaussian process latent variable models

GPLVMs were introduced in the context of visualization of high-dimensional data (Lawrence 2003). GPLVMs perform nonlinear dimensionality reduction in the context of Gaussian processes. The underlying probabilistic model is still a GP regression model as defined in (2). However, the input values $\mathbf{X}$ are not given and become latent variables that need to be determined during learning. In the GPLVM, this is done by optimizing over both the latent space $\mathbf{X}$ and the hyperparameters $\boldsymbol{\theta}_y$:

$$\langle \mathbf{X}^*, \boldsymbol{\theta}_y^* \rangle = \underset{\mathbf{X}, \boldsymbol{\theta}_y}{\operatorname{argmax}} \log \; p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}_y). \tag{8}$$

This optimization can be performed using scaled conjugate gradient descent. In practice, the approach requires a good initialization to avoid local maxima. Typically, such initializations are done via PCA or Isomap (Lawrence 2005; Wang et al. 2008).

The standard GPLVM approach does not impose any constraints on the latent space. It is thus not able to take advantage of the specific structure underlying dynamical systems. Recent extensions of GPLVMs, namely Gaussian Process Dynamical Models (Wang et al. 2008) and hierarchical GPLVMs (Lawrence and Moore 2007), can model dynamic systems by introducing a prior over the latent space $\mathbf{X}$, which results in the following joint distribution over the observed space, the latent space, and the hyperparameters:

$$p(\mathbf{Y}, \mathbf{X}, \boldsymbol{\theta}_y, \boldsymbol{\theta}_x) = p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}_y) p(\mathbf{X} \mid \boldsymbol{\theta}_x) p(\boldsymbol{\theta}_y) p(\boldsymbol{\theta}_x). \tag{9}$$

Here, $p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}_y)$ is the standard GPLVM term, $p(\mathbf{X} \mid \boldsymbol{\theta}_x)$ is the prior modeling the dynamics in the latent space, and $p(\boldsymbol{\theta}_y)$ and $p(\boldsymbol{\theta}_x)$ are priors over the hyperparameters. The dynamics prior is again modeled as a Gaussian process

$$p(\mathbf{X} \mid \boldsymbol{\theta}_x) = \mathcal{N}(\mathbf{X}; 0, \mathbf{K}_x + \sigma_m^2 \mathbf{I}), \tag{10}$$

where $\mathbf{K}_x$ is an appropriate kernel matrix and σ_m is the associated noise term. In Sect. 4, we will discuss different dynamics kernels in the context of learning GP-BayesFilters. The unknown values for this model are again determined via maximizing the log posterior of (9):

$$\begin{aligned} \langle \mathbf{X}^*, \boldsymbol{\theta}_y^*, \boldsymbol{\theta}_x^* \rangle = \underset{\mathbf{X}, \boldsymbol{\theta}_y, \boldsymbol{\theta}_x}{\operatorname{argmax}} \big(& \log p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}_y) \\ & + \log p(\mathbf{X} \mid \boldsymbol{\theta}_x) + \log p(\boldsymbol{\theta}_y) + \log p(\boldsymbol{\theta}_x) \big). \end{aligned} \tag{11}$$

Such extensions to GPLVMs have been used successfully to model temporal data such as motion capture sequences (Lawrence and Moore 2007; Wang et al. 2008) and visual tracking data (Urtasun et al. 2006).

3.3 GP-BayesFilters

GP-BayesFilters are Bayes filters that use GP regression to learn prediction and observation models from training data. Bayes filters recursively estimate posterior distributions over the state $\mathbf{x}_t$ of a dynamical system at time t conditioned on sensor data $\mathbf{z}_{1:t}$ and control information $\mathbf{u}_{1:t-1}$. Key components of every Bayes filter are the prediction model, $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_{t-1})$, and the observation model, $p(\mathbf{z}_t \mid \mathbf{x}_t)$ which are shown in graphical form in Fig. 2. The prediction model describes how the state $\mathbf{x}$ evolves in time based on the control input $\mathbf{u}$. The observation model describes the likelihood of making an observation $\mathbf{z}$ given the state $\mathbf{x}$. In robotics, these models are typically parametric descriptions of the underlying processes; see Thrun et al. (2005) for several examples.

GP-BayesFilters use Gaussian process regression models for both prediction and observation models. Such models can be incorporated into different versions of Bayes filters and have been shown to outperform parametric models (Ko and Fox 2008). Learning the models of GP-BayesFilters requires ground truth sequences of a dynamical system containing for each time step a control command, $\mathbf{u}_{t-1}$, an observation, $\mathbf{z}_t$, and the corresponding ground truth state, $\mathbf{x}_t$. GP prediction and observation models can then be learned based on training data

$$D_p = \langle (\mathbf{X}, \mathbf{U}), \Delta\mathbf{X} \rangle,$$

$$D_o = \langle \mathbf{X}, \mathbf{Z} \rangle,$$

where $\mathbf{X}$ is a matrix containing the sequence of ground truth states, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T-1}]$, $\Delta\mathbf{X}$ is a matrix containing the state changes, $\Delta\mathbf{X} = [\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_2, \ldots, \mathbf{x}_T - \mathbf{x}_{T-1}]$, and $\mathbf{U}$ and $\mathbf{Z}$ contain the sequences of controls and observations, respectively. By plugging these training sets into (5) and (6), one gets GP prediction and observation models. The prediction model maps from a state, $\mathbf{x}_{t-1}$, and a control, $\mathbf{u}_{t-1}$, to change in state, $\mathbf{x}_t - \mathbf{x}_{t-1}$, while the observation model maps from a state, $\mathbf{x}_t$, to an observation, $\mathbf{z}_t$. These probabilistic models can be readily incorporated into Bayes filters such as particle filters and unscented Kalman filters. An additional derivative of (5) provides the Taylor expansion needed for extended Kalman filters (Ko and Fox 2008).

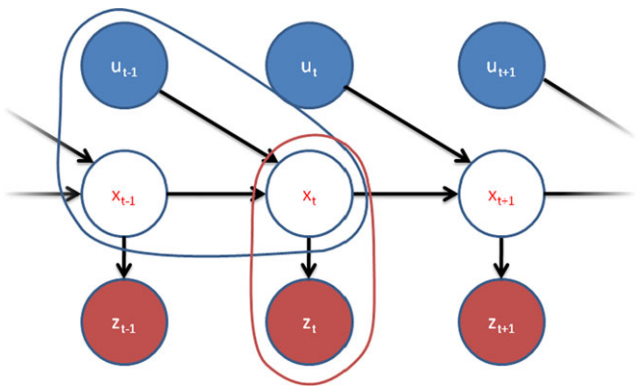

Fig. 2 (Color online) Graphical model of a Bayes filter. *The blue outline* indicates the dynamics model. *The red outline* shows the observation model

The need for ground truth training data is a key limitation of GP-BayesFilters and other applications of GP regression models in robotics. While it might be possible to collect ground truth data using accurate sensors (Ko and Fox 2008; Nguyen-Tuong et al. 2008; Plagemann et al. 2007) or manual labeling (Ferris et al. 2006), the ability to learn GP models based on weakly labeled or unlabeled data significantly extends the range of problems to which such models can be applied.

4 GPBF-Learn

In this section we show how GP-BayesFilters can be learned from weakly labeled data. While the extensions of GPLVMs described in Sect. 3.2 are designed to model dynamical systems, they lack important abilities needed to make them fully useful for robotics applications. First, they do not consider control information, which is extremely important for learning accurate prediction models in robotics. Second, they optimize the values of the latent variables (states) solely based on the output samples (observations) and GP dynamics in the latent space. However, in state estimation scenarios, one might want to impose stronger constraints on the latent space $\mathbf{X}$. For example, it is often desirable that latent states $\mathbf{x}_t$ correspond to physical entities such as the location of a robot. To enforce such a relationship between latent space and physical robot locations, it would be advantageous if one could label a subset of latent points with their physical counterparts and then constrain the latent space optimization to consider these labels.

We now introduce GPBF-Learn, which overcomes limitations of existing techniques. The training data for GPBF-Learn, $D = [\mathbf{Z}, \mathbf{U}, \widehat{\mathbf{X}}]$, consists of time stamped sequences containing observations, $\mathbf{Z}$, controls, $\mathbf{U}$, and weak labels, $\widehat{\mathbf{X}}$, for the latent states. In the context discussed here, the labels provide noisy information about subsets of the latent states. Given training data D, the posterior over the sequence of hidden states and hyperparameters is as follows:

$$p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid \mathbf{Z}, \mathbf{U}, \widehat{\mathbf{X}}) \propto p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z)\, p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x)\, p(\mathbf{X} \mid \widehat{\mathbf{X}})\, p(\boldsymbol{\theta}_z)\, p(\boldsymbol{\theta}_x). \quad (12)$$

In GPBF-Learn, both the observation model, $p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z)$, and the prediction model, $p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x)$, are Gaussian processes, and $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_z$ are the hyperparameters of these GPs. While the observation model in (12) is the same

as in the GPLVM for dynamical systems (9), the prediction GP now includes control information. Furthermore, the GPBF-LEARN posterior contains an additional term for labels, $p(\mathbf{X} \mid \widehat{\mathbf{X}})$, which we describe next.

4.1 Weak labels

The labels $\widehat{\mathbf{X}}$ represent prior knowledge about individual latent states $\mathbf{X}$. For instance, it might not be possible to generate highly accurate ground truth states for every data point in the training set. Instead, one might only be able to provide accurate labels for a small subset of states, or noisy estimates for the states. At the same time, such labels might still be extremely valuable since they guide the latent variable model to determine a latent space that is similar to the desired, physical space. While the form of prior knowledge can take on various forms, we here consider labels that represent independent Gaussian priors over latent states:

$$p(\mathbf{X} \mid \widehat{\mathbf{X}}) = \prod_{\hat{\mathbf{x}}_t \in \widehat{\mathbf{X}}} \mathcal{N}(\mathbf{x}_t; \hat{\mathbf{x}}_t, \sigma^2_{\hat{\mathbf{x}}_t}). \tag{13}$$

Here, $\sigma^2_{\hat{\mathbf{x}}_t}$ is the uncertainty in label $\hat{\mathbf{x}}_t$. As noted above, $\widehat{\mathbf{X}}$ can impose priors on all or any subset of latent states. As we will show in the experiments, these additional terms generate more consistent tracking results on test data.

We now examine use of weak labels at either extreme of very high, or very low uncertainty. With accurate knowledge of the latent states, $\sigma_{\hat{\mathbf{x}}_t}$ becomes very small. As a result, the latent states $\mathbf{X}$ do not move at all during GPBF-LEARN optimization. At this extreme, only hyperparameters are learned, and thus the learning becomes equivalent to simple Gaussian process optimization. On the other hand, if no prior knowledge exists about the latent states, $\sigma_{\hat{\mathbf{x}}_t}$ becomes infinitely large. This essentially gives a uniform distribution over the latent states $\mathbf{X}$. This is very similar to GPDM as now the latent states have complete freedom to move around.

Currently, the label uncertainty $\sigma^2_{\hat{\mathbf{x}}_t}$ is not integrated into the probabilistic framework described by GPBF-LEARN and must be selected independently via either manual tuning, or cross validation.

4.2 GP dynamics models

GP dynamics priors, $p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x)$, do not constrain individual states but model prior information of how the system evolves over time. They provide substantial flexibility for modeling different aspects of a dynamical system. Intuitively, these priors encourage latent states $\mathbf{X}$ that correspond to smooth mappings from past states and controls to future states. Even though the dynamics GP is an integral part of the posterior model (12), for exposure reason it is easier to treat it as if it was a separate GP.

Different dynamics models are achieved by changing the specific values for the input and output data used for this dynamics GP. We denote by $\mathbf{X}^{\text{in}}$ and $\mathbf{X}^{\text{out}}$ the input and output data for the dynamics GP, where $\mathbf{X}^{\text{in}}$ is typically derived from states at specific points in time, and $\mathbf{X}^{\text{out}}$ is derived from states at the next time step. To more strongly emphasize the sequential aspect of the dynamics model we will use time t to index data points. Using the GP dynamics model we get

$$p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x) = \mathcal{N}(\mathbf{X}^{\text{out}}; 0, \mathbf{K}_{\text{x}} + \sigma_x^2 \mathbf{I}), \tag{14}$$

where σ_x^2 is the noise of the prediction model, and the kernel matrix $\mathbf{K}_{\text{x}}$ is defined via the kernel function on input data to the dynamics GP: $\mathbf{K}_{\text{x}}[t, t'] = k(\mathbf{x}^{\text{in}}_t, \mathbf{x}^{\text{in}}_{t'})$, where $\mathbf{x}^{\text{in}}_t$ and $\mathbf{x}^{\text{in}}_{t'}$ are input vectors for time steps t and t', respectively.

The specification of $\mathbf{X}^{\text{in}}$ and $\mathbf{X}^{\text{out}}$ determines the dynamics prior. Consider the most basic dynamics GP which solely models a mapping from the state at time $t-1$, $\mathbf{x}_{t-1}$, to the state at time t, $\mathbf{x}_t$. In this case we get the following specification:

$$\begin{aligned} \mathbf{x}^{\text{in}}_t &= \mathbf{x}_{t-1}, \\ \mathbf{x}^{\text{out}}_t &= \mathbf{x}_t. \end{aligned} \tag{15}$$

Optimization with such a dynamics model encourages smooth state sequences $\mathbf{X}$. Generating smooth *velocities* can be achieved by setting $\mathbf{x}^{\text{in}}_t$ to $\Delta\mathbf{x}_{t-1}$ and $\mathbf{x}^{\text{out}}_t$ to $\Delta\mathbf{x}_t$, where $\Delta\mathbf{x}_t$ represents the velocity $[\mathbf{x}_t - \mathbf{x}_{t-1}]$ at time t (Wang et al. 2008). It should be noted that such a velocity model can be incorporated without adding a velocity dimension to the latent space. A more complex, localized dynamics model that takes control and velocity into account can be achieved by the following settings:

$$\begin{aligned} \mathbf{x}^{\text{in}}_t &= [\mathbf{x}_{t-1}, \Delta\mathbf{x}_{t-1}, \mathbf{u}_{t-1}]^T, \\ \mathbf{x}^{\text{out}}_t &= \Delta\mathbf{x}_t. \end{aligned} \tag{16}$$

This model encourages smooth changes in velocity depending on control input. By adding $\mathbf{x}_{t-1}$ to $\mathbf{x}^{\text{in}}_t$, the dynamics model becomes *localized*, that is, the impact of control on velocity can be different for different states. While one could also model higher order dependencies, we here stick to the one given in (17), which corresponds to a relatively standard prediction model for Bayes filters.

4.3 Optimization

Just as regular GPLVM models, GPBF-LEARN determines the unknown values of the latent states $\mathbf{X}$ by optimizing the log of the posterior over the latent state sequence and the hyperparameters. The log of (12) is given by

$$\log p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid D)$$
$$= \log p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z) + \log p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x)$$
$$+ \log p(\mathbf{X} \mid \widehat{\mathbf{X}}) + \log p(\boldsymbol{\theta}_z) + \log p(\boldsymbol{\theta}_x) + \text{const}, \quad (17)$$

where D represents the training data $[\mathbf{Z}, \mathbf{U}, \widehat{\mathbf{X}}]$. We perform this optimization using scaled conjugate gradient descent (Wang et al. 2008). The gradients of the log are given by:

$$\frac{\partial \log p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid \mathbf{Z}, \mathbf{U})}{\partial \mathbf{X}}$$
$$= \frac{\partial \log p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z)}{\partial \mathbf{X}} + \frac{\partial \log p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x)}{\partial \mathbf{X}}$$
$$+ \frac{\partial \log p(\mathbf{X} \mid \widehat{\mathbf{X}})}{\partial \mathbf{X}}, \quad (18)$$

$$\frac{\partial \log p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid D)}{\partial \boldsymbol{\theta}_x} = \frac{\partial \log p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x)}{\partial \boldsymbol{\theta}_x} + \frac{\partial \log p(\boldsymbol{\theta}_x)}{\partial \boldsymbol{\theta}_x}, \quad (19)$$

$$\frac{\partial \log p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid D)}{\partial \boldsymbol{\theta}_z} = \frac{\partial \log p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z)}{\partial \boldsymbol{\theta}_z} + \frac{\partial \log p(\boldsymbol{\theta}_z)}{\partial \boldsymbol{\theta}_z}. \quad (20)$$

The individual derivatives follow as

$$\frac{\partial \log p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z)}{\partial \mathbf{X}} = \frac{1}{2}\text{trace}\left(\mathbf{K}_Z^{-1}\mathbf{Z}\mathbf{Z}^t\mathbf{K}_Z^{-1} - \mathbf{K}_Z^{-1}\right)\frac{\partial \mathbf{K}_Z}{\partial \mathbf{X}},$$
$$\frac{\partial \log p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z)}{\partial \theta_Z} = \frac{1}{2}\text{trace}\left(\mathbf{K}_Z^{-1}\mathbf{Z}\mathbf{Z}^t\mathbf{K}_Z^{-1} - \mathbf{K}_Z^{-1}\right)\frac{\partial \mathbf{K}_Z}{\partial \boldsymbol{\theta}_z},$$
$$\frac{\partial \log p(\mathbf{X} \mid \boldsymbol{\theta}_x, \mathbf{U})}{\partial \mathbf{X}}$$
$$= \frac{1}{2}\text{trace}\left(\mathbf{K}_X^{-1}\mathbf{X}_{\text{out}}\mathbf{X}_{\text{out}}^T\mathbf{K}_X^{-1} - \mathbf{K}_X^{-1}\right)\frac{\partial \mathbf{K}_X}{\partial \mathbf{X}}$$
$$- \mathbf{K}_X^{-1}\mathbf{X}_{\text{out}}\frac{\partial \mathbf{X}_{\text{out}}}{\partial \mathbf{X}},$$
$$\frac{\partial \log p(\mathbf{X} \mid \boldsymbol{\theta}_x, \mathbf{U})}{\partial \boldsymbol{\theta}_x}$$
$$= \frac{1}{2}\text{trace}\left(\mathbf{K}_X^{-1}\mathbf{X}_{\text{out}}\mathbf{X}_{\text{out}}^T\mathbf{K}_X^{-1} - \mathbf{K}_X^{-1}\right)\frac{\partial \mathbf{K}_X}{\partial \boldsymbol{\theta}_x},$$
$$\frac{\partial \log p(\mathbf{X} \mid \widehat{\mathbf{X}})}{\partial \mathbf{X}[i, j]} = -(\mathbf{X}[i, j] - \widehat{\mathbf{X}}[i, j])/\sigma^2_{\hat{\mathbf{x}}_t},$$

where $\frac{\partial \mathbf{K}}{\partial \mathbf{X}}$ and $\frac{\partial \mathbf{K}}{\partial \theta}$ are the matrix derivatives. They are formed by taking the partial derivative of the individual elements of the kernel matrix with respect to $\mathbf{X}$ or the hyperparameters, respectively.

4.4 GPBF-LEARN algorithm

A high level overview of the GPBF-LEARN algorithm is given in Table 1. The input to GPBF-LEARN consists of training data containing a sequence of observations, $\mathbf{Z}$, control inputs, $\mathbf{U}$, and weak labels, $\widehat{\mathbf{X}}$. In the first step, the unknown latent states $\mathbf{X}$ are initialized using the information provided by the weak labels. This is done by setting every latent state to the estimate provided by $\widehat{\mathbf{X}}$. In the sparse labeling case, the states without labels are initialized by linear interpolation between those for which a label is given. In the fully unsupervised case, where $\widehat{\mathbf{X}}$ is empty, we use N4SID to initialize the latent states (Van Overschee and De Moor 1996). In our experiments, N4SID provides initialization that is far superior to the standard PCA initialization used by Lawrence (2005) and Wang et al. (2008). Then, in Step 2, scaled conjugate gradient descent determines the latent states and hyperparameters via optimization of the log posterior (17). This iterative procedure computes the gradients (18)–(20) during each iteration using the dynamics model and the weak labels. Finally, the resulting latent states $\mathbf{X}^*$, along with the observations and controls are used to learn a GP-BayesFilter, as described in Sect. 3.3.

Table 1 The GPBF-LEARN algorithm

Algorithm GPBF-LEARN $(\mathbf{Z}, \mathbf{U}, \widehat{\mathbf{X}})$**:**

1: *if* $(\widehat{\mathbf{X}} \neq \emptyset)$
  $\mathbf{X} := \widehat{\mathbf{X}}$
  else
  $\mathbf{X} := \text{N4SID}_x(\mathbf{Z}, \mathbf{U})$
2: $\langle \mathbf{X}^*, \boldsymbol{\theta}_x^*, \boldsymbol{\theta}_z^* \rangle := \text{SCG\_optimize}\left(\log p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid \mathbf{Z}, \mathbf{U}, \widehat{\mathbf{X}})\right)$
3: **GPBF** $:= \text{Learn\_gpbf}(\mathbf{X}^*, \mathbf{U}, \mathbf{Z})$
4: *return* **GPBF**

In essence, the final step of the algorithm "compiles" the complex latent variable model into an efficient, online GP-BayesFilter. The key difference between the filter model and the latent variable model is due to the fact that the filter model makes a first order Markov assumption. The latent variable model, on the other hand, optimizes all latent points jointly and these points are all correlated via the GP kernel matrix. To reflect the difference between these models, we learn new hyperparameters for the GP-BayesFilter. This final step of the algorithm also allows an opportunity for use of more sophisticated Gaussian process models. For example, sparse GPs or heteroscedastic GPs (Kersting et al. 2007), ones where the noise predictions are state dependent, can be used at this time.

4.5 Time alignment

We now show how time alignment for time series data can be realized using the GPBF-LEARN framework. The capability to do this alignment is an important property of GPBF-LEARN. For 1D time alignment, we are interested in learning a one-to-one mapping between latent states and time indices of an episode. In this context, an episode is one of a

series of similar events performed by a system. For example, a lap around the track by the slotcar represents an episode. To achieve this, two conditions are necessary. First, just like loop closing in robotic SLAM (Thrun et al. 2005), latent states for different episodes must correspond to the same time index. Second, two latent states within a single episode must not map to the same time index. The first condition, the alignment of multiple episodes, happens automatically as part of the optimization process. This is because distinct observations are used to help align across different episodes. The dynamics models then help fill in the gaps where observations are indistinct. To achieve the second condition, the 1D latent state space must be monotonically increasing for each episode. To note, simply using the more complex dynamics model which incorporates velocities is not enough to guarantee monotonically increasing latent states, since negative velocities are not prohibited by the model.

Before we go further into detail the details of time alignment, the nomenclature must be slightly revised in order to accommodate episodic data. The log posterior formula remains essentially the same. Now, $\mathbf{x}_t^k$ denotes the 1D latent state of episode k at time t. The total number of timesteps for episode k is denoted by T^k. $\mathbf{X}$ is now a concatenation of all $\mathbf{X}^k$.

For the case of GPBF-LEARN, in order to have $\mathbf{X}^k$ monotonically increasing, the change in $\mathbf{X}^k$ must be greater than 0. That is, $\forall t, k : \Delta\mathbf{x}_t^k > 0$. To achieve this, we constrain velocities to take on the form of $\Delta\mathbf{x}_t^k = e^{w_t^k}$, where w_t^k is a new parameter and $\mathbf{W}^k = [w_1^k, w_2^k, \ldots, w_{T^k}^k]$. The correspondence between $\mathbf{X}^k$ and $\mathbf{W}^k$ is

$$\mathbf{x}_t^k = \sum_{i=1}^{t} e^{w_i^k}. \tag{21}$$

Note that every episode is assumed to start at latent state 0 as a by-product of this equation. In addition, the end of one episode and the beginning of the next are not connected probabilistically within the GPBF-LEARN framework.

We now reparameterize the original GPBF-LEARN optimization formula. Instead of solving for $\mathbf{X}$, we now solve for $\mathbf{W}$. (Similar to $\mathbf{X}$, $\mathbf{W}$ is a concatenation of all $\mathbf{W}^k$.)

A new log posterior formula can be defined based on $\mathbf{W}$. The first step of this new formula is to extract $\mathbf{X}$ from $\mathbf{W}$. The log posterior calculation then proceeds as in the original. The old and new formulas are denoted as follows,

$$L = \log p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid D), \tag{22}$$

$$L_{\text{new}} = \log p(\mathbf{W}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid D). \tag{23}$$

The derivatives of the optimization formula are also affected by this reparameterization. Instead of derivatives with respect to $\mathbf{X}$, we now have to take the derivative with respect to $\mathbf{W}$. The derivatives of L_{new} are very closely related to the derivatives of L. The derivative of w_t depends on every derivative of $\mathbf{x}_t$ that contains w_t. Using the chain rule,

$$\frac{\partial L_{\text{new}}}{\partial w_t^k} = \frac{\partial L}{\partial \mathbf{x}_1^k}\frac{\partial \mathbf{x}_1^k}{\partial w_t^k} + \frac{\partial L}{\partial \mathbf{x}_2^k}\frac{\partial \mathbf{x}_2^k}{\partial w_t^k} + \cdots + \frac{\partial L}{\partial \mathbf{x}_t^k}\frac{\partial \mathbf{x}_t^k}{\partial w_t^k}, \tag{24}$$

which reduces to

$$\frac{\partial L_{\text{new}}}{\partial w_t^k} = \sum_{i=1}^{t} \frac{\partial L}{\partial \mathbf{x}_i^k} e^{w_t^k}. \tag{25}$$

The derivatives of the hyperparameters $\frac{\partial L_{\text{new}}}{\partial \theta_X}$ and $\frac{\partial L_{\text{new}}}{\partial \theta_Z}$ remain the same.

Latent states for each episode are monotonically increasing after optimization. We show how 1D time alignment can be performed for the slotcar system in our experiments. Although this 1D time alignment may not be sufficient for tracking complex systems, we believe it can be used for time alignment in a host of problems. This technique can be augmented by adding other unconstrained dimensions to the latent space if one dimension is not enough. This technique has some advantages over other traditional dynamic time warping algorithms as it is able to handle multidimensional observation data in a fully probabilistic framework. A naive extension of the method described in Rabiner et al. (1978) for multidimensional observations would at least require tuning separate weights for each observation dimension.

4.6 Trajectory replay via GPBF-LEARN

In this section, we propose a method for replay of trajectories based on expert demonstrations using the GPBF-LEARN framework. Specifically, given one or a series of demonstrations by a human expert, we want to learn how to control the system in the same manner.

At a high level, in order to do trajectory replay, the latent states and GP models are learned using GPBF-LEARN. There is a corresponding control used for each point in the latent space $\mathbf{X}$ based on the time index of both the control input and the learned latent state. The idea is to build a mapping from latent space to controls. This relationship is outlined in the graphical model shown in Fig. 3. In this case, we use a Gaussian process to learn the mapping. Again, as for the dynamics models, different control models are possible. The input for the control model can use just the latent states $\mathbf{X}$, or incorporate velocities. The simplest model will yield a GP with training data:

$$D_c = \langle \mathbf{X}, \mathbf{U} \rangle. \tag{26}$$

In order to do the actual trajectory replay, the learned GP models are used to track within the latent space via GP-BayesFilter. Given the estimated latent state, an estimated

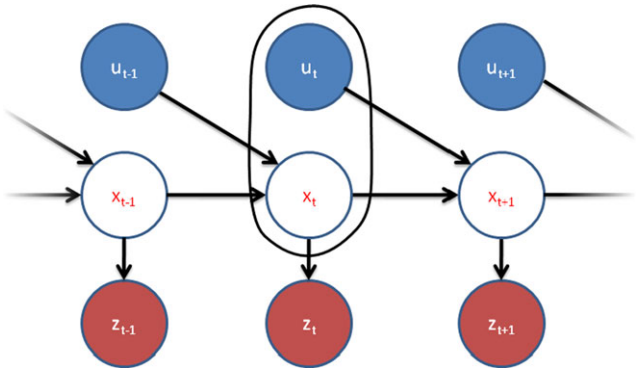

Fig. 3 (Color online) Graphical model showing the relationship of inputs and outputs for the control model

control is recovered from the GP control model. This control is then sent to the system. We show how this technique can be used to replay trajectories in the slotcar problem. Note that this method for trajectory replay is rather simplistic, but it could be readily extended to integrate more sophisticated online optimization such as receding horizon control used in Coates et al. (2008).

This control model works for the slotcar where drift is not so much of a problem because the car is fixed to the track. However, for more complex systems, the generated controls can lead the system outside the manifold described by the training data, at which point tracking might fail. The simple control model described above has no mechanism to correct for such drift. However, if there was a way to find "good" controls that tend to reduce drift, we can learn control models based only on these controls. The resulting control model avoids leaving parts of the state space covered well by the training data. The challenge is to determine the amount of drift introduced by a particular control since it may not be directly calculable. For example, calculating drift in observation space may not yield the correct value due to aliasing issues. Two similar observations may not derive from similar latent states. The key idea is to test the influence of each control in the original training data of the control model. If the control leads to control predictions which cause the dynamics model to become more uncertain, then we remove that control from the control model.

The process for extracting the control training data is shown in pseudocode in Table 2. This is shown for the simplest dynamics model with $D_p = \langle (\mathbf{X}, \mathbf{U}), \mathbf{X}' \rangle$ where $\mathbf{X}' = [\mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_T]$ is the latent data offset by one time step. In Step 1, we initialize the control training data as empty sets. Then, in Step 2, we loop over all the training pairs $\langle x_t, u_t \rangle$ which make up the control model training data. In Step 3, we predict the control under two scenarios. In the first, we assume that the current training pair is part of the training data. In the second, we test the model without that training pair. New latent state predictions are then made using the dynamics model with each predicted control in Step 4. Step 5 tests the prediction uncertainty of these new states. Finally, in Step 6, the considered training pair is only accepted if it results in a less uncertain control prediction one time step in the future.

Table 2 Algorithm for selecting training data for the advanced control model

Algorithm SelectTrainingData $(\mathbf{X}, \mathbf{U}, \mathbf{X}')$**:**

1: $\mathbf{X}^{\text{new}} = \mathbf{U}^{\text{new}} = \emptyset$

2: *for t from* 1 *to* $T - 1$

3: $\mathbf{u}^{\text{w/}} = \text{GP}_{\mu}\left(\mathbf{x}_t, \langle \mathbf{X}, \mathbf{U} \rangle\right)$
$\mathbf{u}^{\text{w/o}} = \text{GP}_{\mu}\left(\mathbf{x}_t, \langle \mathbf{X} - \mathbf{x}_t, \mathbf{U} - \mathbf{u}_t \rangle\right)$

4: $\mathbf{x}^{\text{w/}} = \text{GP}_{\mu}\left((\mathbf{x}_t, \mathbf{u}^{\text{w/}}), \langle (\mathbf{X}, \mathbf{U}), \mathbf{X}' \rangle\right)$
$\mathbf{x}^{\text{w/o}} = \text{GP}_{\mu}\left((\mathbf{x}_t, \mathbf{u}^{\text{w/o}}), \langle (\mathbf{X}, \mathbf{U}), \mathbf{X}' \rangle\right)$

5: $\sigma^{\text{w/}} = \text{GP}_{\Sigma}\left((\mathbf{x}^{\text{w/}}, \mathbf{u}_{t+1}), \langle (\mathbf{X}, \mathbf{U}), \mathbf{X}' \rangle\right)$
$\sigma^{\text{w/o}} = \text{GP}_{\Sigma}\left((\mathbf{x}^{\text{w/o}}, \mathbf{u}_{t+1}), \langle (\mathbf{X}, \mathbf{U}), \mathbf{X}' \rangle\right)$

6: *if* $(\sigma_{\text{w/}} < \sigma_{\text{w/o}})$
insert $\mathbf{x}_t$ *into* $\mathbf{X}^{\text{new}}$
insert $\mathbf{u}_t$ *into* $\mathbf{U}^{\text{new}}$

7: *return* $\mathbf{X}^{\text{new}}$, $\mathbf{U}^{\text{new}}$

A new control model is constructed using this data. This new set of training data can be used directly with the previously found hyperparameters. This algorithm is also written assuming the dynamics model has a 1D output. We assume independence between multiple outputs, so the final uncertainty would just be the product of the uncertainties of the individual dimensions for a multi-output dynamics model.

Because of the binary nature of this pruning approach, care must be taken to avoid losing all controls in regions of latent space. At this point in time, we avoid this by having enough variability in the demonstration trajectories so that no region of the latent space will have controls which only increase uncertainty. Future work may explore a probabilistic approach to pruning.

5 Experiments

In this section, we evaluate the use of GPBF-LEARN under a variety of conditions. In the first part, GPBF-LEARN is analyzed for conditions where prior knowledge of the latent states is available, either through noisy, or sparsely labeled data. GPBF-LEARN is then tested as a method for system identification where no prior knowledge of the state is available. Under this scenario, we show how the system can be tracked in the latent space via a GP-BayesFilter extracted from the learned latent states. GPBF-LEARN is also compare to a state-of-the-art subspace identification algorithm. The third part demonstrates some unique features enabled

by the GPBF-LEARN framework. We show how time alignment of episodic behavior can be obtained, and how system control can be achieved using models found by GPBF-LEARN. The experiments are performed on two very different test platforms. The first is toy car, and the second is a robotic arm. These two systems will be described in line with their respective experiments.

In an additional experiment not reported here, we compared the two dynamics models described in Sect. 4.2. Using 10-step ahead prediction as evaluation criteria, we found that our control and velocity based model (16) significantly outperforms the simpler model (15) that is typically used for GPLVMs. In fact, our model reduces the prediction error by almost 45%, from 29.2 to 16.1 cm. We use this more complex model in all our tests.

5.1 Incorporating noisy and spare labels

In this section we demonstrate that GPBF-LEARN can learn a latent (state) space $\mathbf{X}$ that is consistent with a desired latent space specified via weak labels $\widehat{\mathbf{X}}$. Here, the desired latent space is the 1D position of the car along the track. In this scenario, we assume that the training data contains noisy or sparse labels $\widehat{\mathbf{X}}$. First, we will describe the test platform used.

The experimental setup consists of a track and a miniature car which is guided along the track by a groove, or slot, cut into the track. The left panel in Fig. 1 shows the track, which contains elevation changes as well as banked curves with a total length of about 14 m. An overhead camera is used to track the car. The car can be extracted from the video stream using background subtraction. This information is then combined with a detailed model of the track to provide the car position from the start of the lap at each point in time. These 1D car positions serve as ground truth data for the experiments. The car is a standard 1:32 scale model manufactured by Carrera® International and augmented with an inertial measurement unit (IMU), as shown in the second panel in Fig. 1. The IMU is the InertiaDot sensor developed by colleagues at Intel Research Seattle. It provides 3-axis accelerometer, and 3-axis gyro information and is powered by a small lithium battery. The overall package is quite small at 3 cm by 3 cm by 1 cm and weighs just 10 grams. The gyro measures changes in orientation and provides essentially a turning rate for each axis. These measurements are sent off-board in real-time via a Bluetooth interface. Control signal to the car are supplied by an off-board computer. The controls signal is directly proportional to the amperage supplied the car motor. It is a unitless measure stored as an 8-bit unsigned integer which gives possible values from 0 to 255. The car requires fairly precise control inputs to operate as there are portions of the track that must be driven though quickly, and other portions slowly. The car exhibits

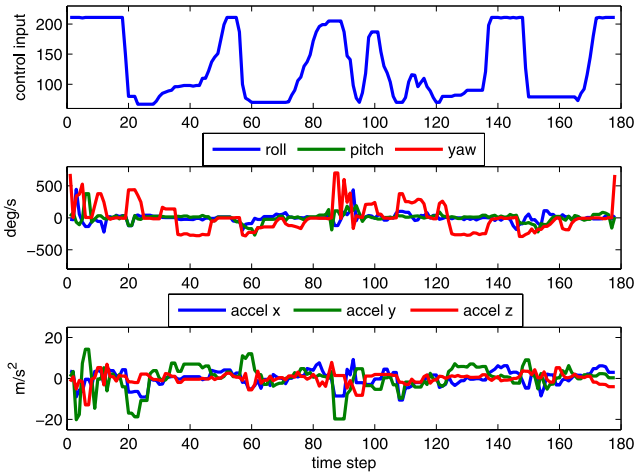

Fig. 4 (Color online) The control inputs (*top*), IMU turning rate in roll, pitch, and yaw (*middle*), and IMU accelerometer observations (*bottom*) for the same run. Shown is data collected over two laps around the track

two main failure modes. It can crash by entering a turn too quickly, or by driving along the banked portions of the track too slowly.

As can be seen in the middle and bottom panel in Fig. 4, both the accelerometer and gyro outputs contain significant noise. In addition, the observation data also includes substantial amounts of aliasing, in which the same measurements occur at many different locations on the track. For instance, all angle differences are close to zero whenever the car moves through a straight section of the track. The observation noise and aliasing makes this problem challenging, and the learning a model of the latent space a necessity since there is no simple unique mapping between the observations $\mathbf{Z}$ and the states $\mathbf{X}$.

In all experiments, GP-UKFs are used to generate tracking results (Ko et al. 2007). Different types of GP-Bayes-Filter such as GP-EKF can be used, but GP-UKF gives a good tradeoff between accuracy and speed.

5.1.1 *Noisy labels*

In this first experiment, we consider the scenario in which one is not able to provide extremely accurate ground truth states for the training data. Instead, one can only provide noisy labels $\widehat{\mathbf{X}}$ for the states. We evaluate four possible approaches to learning a GP-BayesFilter from such data. The first, called INIT, simply ignores the fact that the labels are noisy and learns a GP-BayesFilter using the initial data $\widehat{\mathbf{X}}$. The next two use the noisy labels to initialize the latent variables $\mathbf{X}$, but performs optimization *without* the weak label terms described in Sect. 4.1. We call this approach GPDM, since it results from applying the model of Wang et al. (2008) to this setting. We do this with and without

the use of control data $\mathbf{U}$ in order to distinguish the contributions of the various components. Finally, GPBFL denotes our GPBF-LEARN approach that considers the noisy labels during optimization.

The system state in this scenario is the 1D position of the car along the track, that is, the approach must learn to project the 6D IMU observations $\mathbf{Z}$ along with the control information $\mathbf{U}$ into a 1D latent space $\mathbf{X}$. Training data consists of 5 manually controlled cycles of the car around the track. We perform cross-validation by applying the different approaches to four loops and testing tracking performance on the remaining loop. The overhead camera provides fairly accurate 1D track position. To simulate noisy labels, we added different levels of Gaussian noise to the camera based 1D track locations and used these as $\widehat{\mathbf{X}}$. For each noise level applied to the labels we perform a total of 10 training and test runs. For each run, we extract GP-BayesFilters using the resulting optimized latent states $\mathbf{X}^*$ along with the controls and IMU observations. The quality of the resulting models is tested by checking how close $\mathbf{X}^*$ is to the ground truth states provided by the camera, and by tracking with a GP-UKF on previously unseen test data.

The top panel in Fig. 5 shows a plot of the differences between the learned hidden states, $\mathbf{X}^*$, and the ground truth for different values of noise applied to the labels $\widehat{\mathbf{X}}$. As can be seen, GPBFL is able to recover the correct 1D latent space even for high levels of noise. GPDM which only considers the labels by initializing the latent states generates a high error. This is due to the fact that the optimization performed GPDM lets these latent states "drift" from the desired values. The optimization performed by GPDM without control is even higher than that with control. GPDM without control ends up overly smooth since it does not have controls to constrain the latent states. Not surprisingly, the error of INIT increases linearly in the noise of the labels, since INIT uses these labels as the latent states without any optimization.

The middle panel in Fig. 5 shows the RMS error when running a GP-BayesFilter that was extracted based on the learned hidden states using the different approaches. For clarity, we only show the averages over those runs that did not produce a tracking error. A run is considered a failure if the RMS error is greater than 70 cm. Out of its 80 runs, INIT produced 18 tracking failures, GPDM without controls 11, GPDM with controls 7, while our approach GPBFL produced only one failure. Note that a tracking failure can occur due to both mis-alignment between the learned latent space and high noise in the observations.

As can be seen in the figure, GPBFL is able to learn a GP-BayesFilter that maintains a low tracking RMS error even when the labels $\widehat{\mathbf{X}}$ are very noisy. On the other hand, simply ignoring noise in labels results in increasingly bad tracking performance, as shown by the graph for INIT. In addition, GPDM generates significantly poorer tracking performance than our approach.

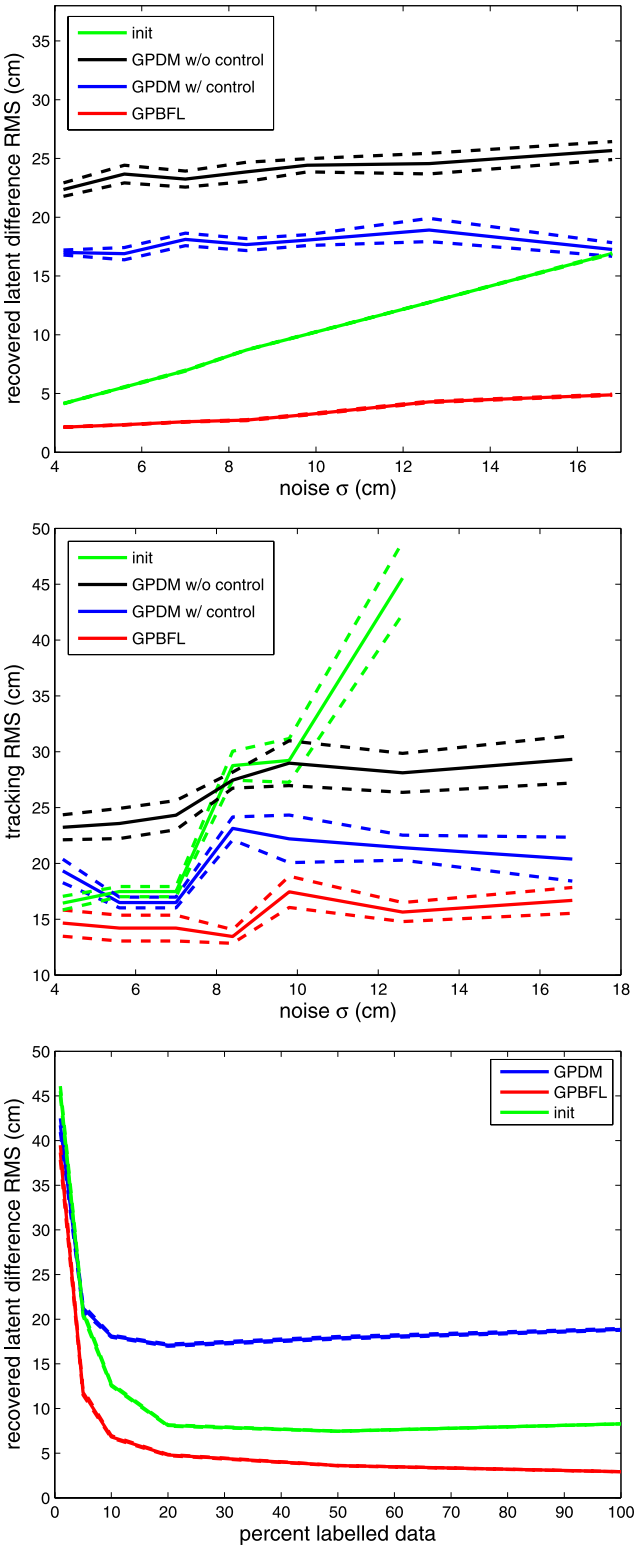

Fig. 5 (Color online) Evaluation of INIT, GPDM, and GPBFL on noisy and sparse labels. *Dashed lines* provide 95% confidence intervals. (*Top*) Difference between the learned latent states $\mathbf{X}^*$ and ground truth as a function of noise level in the labels $\widehat{\mathbf{X}}$. (*Middle*) Tracking errors for different noise levels. (*Bottom*) Difference between the learned latent states and ground truth as a function of label sparsity

5.1.2 Sparse labels

In some settings it might not be possible to provide even noisy labels for all training points. This situation could arise from a sensor that can only capture the robot or object's position in the latent space at much lower frequency than other sensors, or if it can only capture latent states for a fraction of the state space. Here we simulate this scenario by randomly removing noisy labels from the training data. For the approach INIT we generated full labels by linearly interpolating between the sparse labels. The bottom panel in Fig. 5 shows the errors between ground truth 1D latent space and the learned latent space, $\mathbf{X}^*$, for different levels of label sparsity. Again, our approach, GPBFL, learns a more consistent latent space as GPDM, which uses the labels only for initialization. The linear interpolation approach, INIT, outperforms GPDM since the states X do not change at all and thereby avoids drifting from the provided labels.

5.2 GPBF-LEARN for system identification

The next set of experiments demonstrate how GPBF-LEARN can learn models without any labeled data. Here, the training input consists solely of IMU observations $\mathbf{Z}$ and control inputs $\mathbf{U}$. We perform GPBF-LEARN using a training set comprising of about 1400 data points (observation and control pairs), or about 16 laps of data. A very high $\sigma_{\hat{x}_t}$ is used, essentially providing no knowledge about the structure of the latent space. These experiments are broken up into three parts. First, we show how a 3D latent space can be learned by GPBF-LEARN initialized with N4SID, a standard system identification technique. We show how the derived GP-BayesFilter can then be used to track in the latent space. Both GPBF-LEARN and N4SID are then compared in terms of tracking and prediction performance. This section ends with a comparison of GPBF-LEARN with a state of the art system identification method.

5.2.1 Learning the latent states

This first experiment shows how we can learn a 3D latent space with no prior knowledge of the latent space for the slotcar system. A three dimensional latent space is used to encode knowledge about the underlying system. A different number of dimensions could be used, but three dimensions trades off accuracy for ease of presentation. Overall, this is an extremely challenging task for latent variable models. To see, we initialized the latent state of GPBF-LEARN using PCA, as is typically done for GPLVMs (Lawrence 2005; Urtasun et al. 2006; Wang et al. 2008). The latent state is initialized using three principal components to reconstruct the 6D observations from the slotcar IMU. In this case, GPBF-LEARN was not able to learn a smooth model of the latent space. This is because PCA is linear and does not take the dynamics in latent space into account, and thus the GPBF-LEARN optimization is unable to overcome the poor initialization.

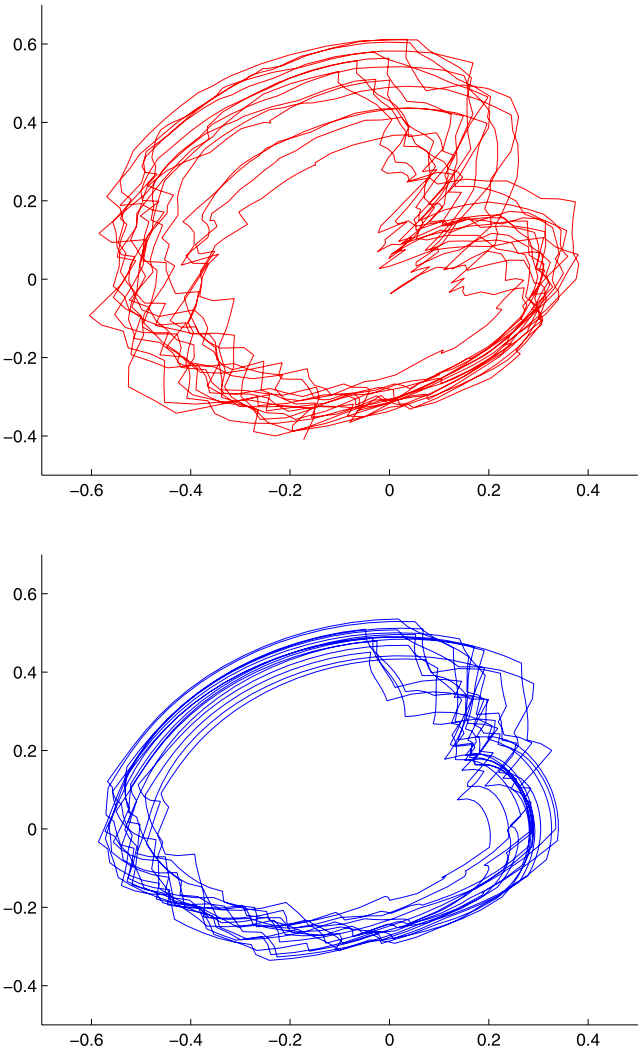

Fig. 6 First two dimensions of a 3D latent space learned by N4SID (*top*) and GPBF-LEARN after optimization (*bottom*)

A different approach for initialization is N4SID, which is a well known linear model for system identification of dynamical systems. A brief description of the algorithm is provided in Appendix. A more thorough treatment can be found here (Van Overschee and De Moor 1996). N4SID provides an estimate of the hidden state which does take into account the system dynamics. This approach determines the matrices from which the hidden state space can then be recovered. The latent space $\mathbf{X}_{\text{N4SID}}$ recovered by N4SID is shown in red in the top panel in Fig. 6. N4SID is able to capture the overall cyclic nature of the track when using an appropriately long time horizon (70 in this case). The time horizon is the number of future and past observations considered in the data matrix. Typically, the longer the time horizon, the

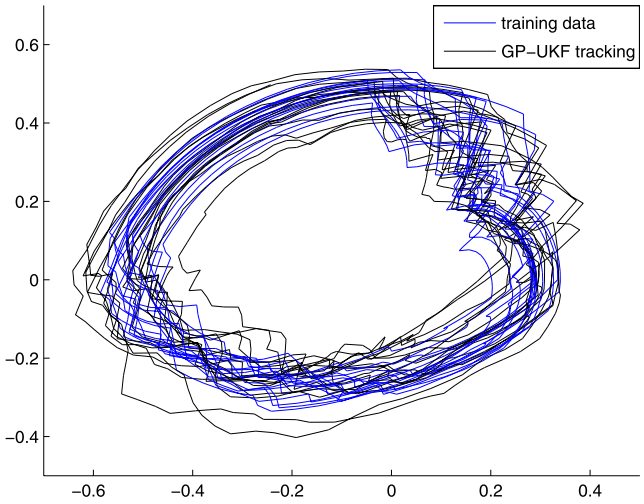

Fig. 7 (Color online) Result of tracking in latent space using GP-UKF (*black*). The original training latent states (*blue*)

smoother the model. However, the recovered latent space is still not very smooth, owing to the linear nature of the model. The main advantage of N4SID is the relative simplicity of the linear dynamics and observation models. Filtering can be done very efficiently using a Kalman filter derived from the N4SID model. Running GPBF-LEARN latent space optimization on the data initialized with N4SID gives us the blue graph shown in the same figure. GPBF-LEARN takes advantage of its underlying non-linear GP model to recover a smooth latent space. The remaining roughness indicates the abruptness of the control inputs and is a desired feature. Note that we would not expect all cycles through the track to be mapped exactly on top of each other, since the slot-car has very different observations depending on its velocity at that track position. The encoding or meaning of the different dimensions of the state space may not be necessarily obvious.

5.2.2 *Tracking in latent states*

We now show the ability to track in the learned latent space. This is done by tracking with a GP-UKF extracted from the optimized training data. The training data consists of approximately 1400 data points. We test by filtering on a previously unseen dataset of approximately the same size. Figure 7 shows GP-UKF tracking within the latent space. The tracking trajectory is indicated in black with the training latent data in blue. The tracking in latent space is not as smooth as the training data because of the observation corrections. However, the tracking trajectory generally does stay within the manifold described by the training data.

We now perform a more thorough comparison of the predictive power of the GPBF-LEARN and N4SID models. To do this, N4SID and GPBF-LEARN models are learned using the previous training data. We compare the performance

Table 3 The Kalman filter algorithm used for evaluating N4SID performance

Algorithm Kalman Filter $(\mathbf{x}_{t-1}, \Sigma_{t-1}, z_t, u_t)$**:**

1: $\bar{\mathbf{x}}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t$
2: $\bar{\Sigma}_t = \mathbf{A}\Sigma_{t-1}\mathbf{A}^{\mathbf{T}} + \mathbf{Q}$
3: $\hat{z}_t = \mathbf{C}\bar{\mathbf{x}}_t$
4: $\mathbf{S}_t = \mathbf{C}\bar{\Sigma}_t\mathbf{C} + \mathbf{R}$
5: $\mathbf{x}_t = \bar{\mathbf{x}}_t + \mathbf{K}(z_t - \hat{z}_t)$
6: $\Sigma_t = (I - \mathbf{K}\mathbf{C})\bar{\Sigma}_t$
7: *return* $\mathbf{x}_t, \Sigma_t, \hat{z}_t, \mathbf{S}_t$

Table 4 Observation prediction quality

Method	RMS accel (m/s^2)	RMS gyros (deg/s)	MLL
Mean	8.56 ± 0.23	239.01 ± 6.94	N/A
N4SID	7.53 ± 0.20	194.94 ± 4.86	1.54 ± 0.19
GPBFL	6.87 ± 0.19	157.75 ± 4.63	3.20 ± 0.22

of filtering based on N4SID and GPBF-LEARN using the same previous test data. For N4SID, the filter used is a standard linear Kalman filter which is described in Table 3.

The temporary variables $\bar{\mathbf{x}}_t$ and $\bar{\Sigma}_t$ are the predicted state mean and covariance based only on the dynamics model. The predicted observation mean and covariance, $\hat{z}_t$ and $\mathbf{S}$ can be computed from these temporary predictions. $\mathbf{x}_t$ and Σ_t are the final predicted state mean and covariance after the observation correction Steps 5 and 6. The matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{K}$ are given by the N4SID algorithm. The noise covariances $\mathbf{R}$ and $\mathbf{Q}$ can be calculated from the observation data and the reconstructed states $\mathbf{X}_{\text{N4SID}}$, respectively.

Since these two techniques exist within two different latent spaces, comparing the filters' ability to track within their respective latent spaces would not be informative. The only true point of comparison is the ability to predict observations. The root mean square (RMS) error between the predicted and actual observation is one natural measure of performance. The mean log-likelihood (MLL) of the actual observation given the predicted observation and predicted observation covariance is another measure.

We run the N4SID-derived and GPBF-LEARN-derived filter on the test data. The RMS observation error and mean log-likelihood are computed after allowing for 100 steps of filter "burn-in". The results are shown in Table 4 and clearly reflect the superior performance of the GPBF-LEARN filter. The row "Mean" represents using the mean of the training observations as the prediction at every time step. This is the baseline for worst-case performance. "Mean" does not have an entry for MLL, since there is no observation prediction covariance to compute the log-likelihood.

In addition to instantaneous observation predictions, the filter framework can be used to make observation predic-

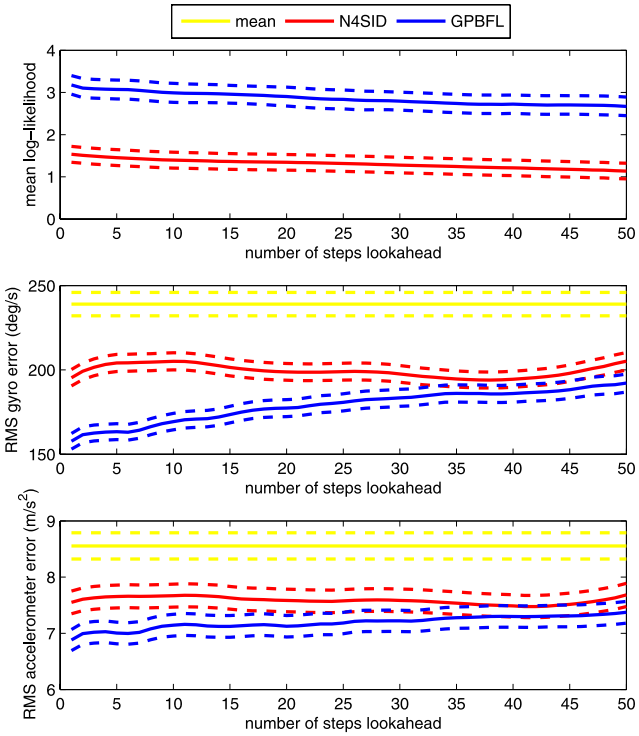

Fig. 8 (Color online) Mean log-likelihood and RMS error of GPBF-LEARN and N4SID for different number of steps of lookahead. To note, 95% error bars are indicated by the *dashed lines*

tions further into the future. That is, the filter can be run up to a certain point, then future predictions can be made by running the filter without observation corrections. For the linear Kalman filter, that means not performing Steps 5 and 6 of the Kalman filter algorithm and taking $\bar{\mathbf{x}}_t$ and $\bar{\Sigma}_t$ as the new state mean and covariance. Observation corrections can be removed from GP-UKF in a similar fashion. RMS error and MLL can be computed as before for different number of steps of lookahead, that is, different number of steps without observation correction. The results are captured in Fig. 8. Note that RMS error of "Mean" does not change since neither the mean training observation or the test observations change with increased lookahead. Performance degrades as the filters predict further into the future. For GPBF-LEARN, it is a result of both an increase in prediction error as well as increase in observation prediction covariance. GPBF-LEARN still has relatively good performance even up to 50 steps of lookahead which represents more than halfway around the track.

5.2.3 *Comparison with kernel CCA method*

In the last experiment of this section, we compare GPBF-LEARN to a state-of-the-art non-linear subspace identification method. The method is the kernel canonical correlation analysis subspace identification algorithm described

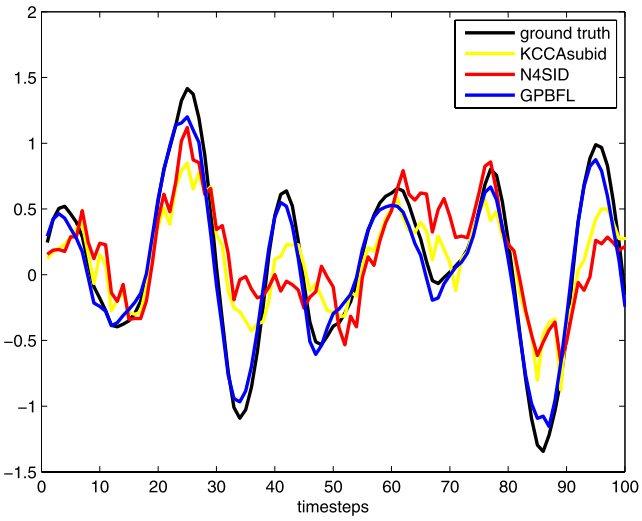

Fig. 9 (Color online) Output after filtering using GPBF-LEARN, the kernel CCA subspace identification method, and N4SID

in Kawahara et al. (2007), herein abbreviated as KCCA-SUBID. The idea underlying the CCA approach to system identification is to use the canonical correlations between past and future observations to find a low dimensional latent state $\mathbf{X}$. This latent state preserves the maximum amount of information from past observations and inputs necessary to predict future observations. This is similar in spirit to how a low dimension representation of data can be reconstructed using PCA. The method under consideration replaces CCA with kernel CCA, thereby generalizing the approach to non-linear system identification.

The system we wish to model is the small problem presented in Kawahara et al. (2007) and Verdult et al. (2004). It is a non-linear system with system dynamics of the form:

$$\dot{x}_t = y_t - 0.1\cos(x_t)(5x_t - 4x_t^3 + x_t^5) - 0.5\cos(x_t)u_t,$$
$$\dot{y}_t = -65x_t + 50x_t^3 - 15x_t^5 - y_t - 100u_t.$$

The output of this system is $\mathbf{y}$ and the input $\mathbf{u}$ is a zero-order-hold white noise evenly distributed between -0.5 and 0.5. The system states are found by solving the ODE using the standard Runge-Kutta method with a step size of 0.05. N4SID, KCCASUBID, and GPBF-LEARN are evaluated on this data. We use one dimensional latent states for all methods, since higher order models result in virtually no error, even for the N4SID model. Both N4SID and GPBF-LEARN use 600 data points for training, and 100 points for testing. KCCASUBID requires tuning of multiple parameters. Therefore, 500 points are used for training, 100 points for validation, and 100 points for testing. The parameters are tuned to minimize the squared error on the validation data. These parameters are then used for testing.

The results of filtering using models built with these three methods are shown in Fig. 9. N4SID and KCCA-

SUBID both run a linear Kalman filter. KCCASUBID operates on kernel matrices, thereby enabling non-linear performance. The GPBF-LEARN method is evaluated with a GP-UKF.

The RMS errors for this plot are 8.08, 6.42, and 2.09 for N4SID, KCCASUBID, and GPBF-LEARN, respectively. This result for KCCASUBID is in line with those previously published (Kawahara et al. 2007). Although the kernel CCA subspace identification is non-linear due to the kernel trick, it does not seem to be able to match the accuracy of GPBF-LEARN for the same number of latent dimensions.

Effort was made to compare KCCASUBID to GPBF-LEARN in the slotcar and arm experimental data sets, but we encountered difficulty in extending the work to multiple dimensions. This requires the tuning of more parameters, and we were unable to find a stable solution. Although faster than GPBF-LEARN, KCCASUBID is still a kernel method, and thus has similar issues with time and space inefficiencies.

5.3 Unique features of GPBF-LEARN

In this section, we explore some interesting problems that can be solved within the GPBF-LEARN framework. The first is obtaining time alignment using a 1D latent dimension. We then show how trajectory replay can be accomplished by building GP control models after learning the latent states. This is demonstrated with both the simple and advanced control model described in Sect. 4.6.

5.3.1 Time alignment in 1D

A unique aspect of GPBF-LEARN is the ability to learn a proper alignment of the data where each portion of the true state corresponds to a similar latent state. This experiment shows how explicit time alignment of the data can be achieved for the slotcar system. For this experiment, each lap around the track is treated as a separate episode. Ideally, after optimization, the recovered latent states would have for every episode, the same point in latent space mapping to a specific point on the track. The monotonic increase in the 1D latent space is then a necessity since we do not want a point in latent space to represent multiple points on the 1D track. We use the GPBF-LEARN formulation described in Sect. 4.5 which enforces monotonically increasing latent states. The GPBF-LEARN optimization problem is initialized by assuming constant velocity from the start to the end. That is, if the episode takes T time steps, the car is assumed to move $1/T$ units in the 1D latent space per time step. As indicated by (21), every episode starts at latent state 0.

The overall alignment of the data can be tested by plotting recovered latent positions for different episodes against the 1D ground truth data. Figure 10 shows the mapping between latent states and the ground truth car position of each episode. The initial alignment is shown in red with each line representing a different episode. One can see from the initialization data that the same 1D latent state maps to very different positions on the track, which is due to the fact that the car did not move at identical velocities in the different episodes. The maximum deviation occurs at latent state 0.737 with 254.8 cm deviation. That means, using the initial alignment, one could be up to 254.8 cm off if one used the 1D latent state as a surrogate for track position. On the other hand, after GPBF-LEARN optimization, the alignment is much tighter. The deviation now is only 47.0 cm or about 3.3% of the track length. GPBF-LEARN gives excellent alignment of the data.

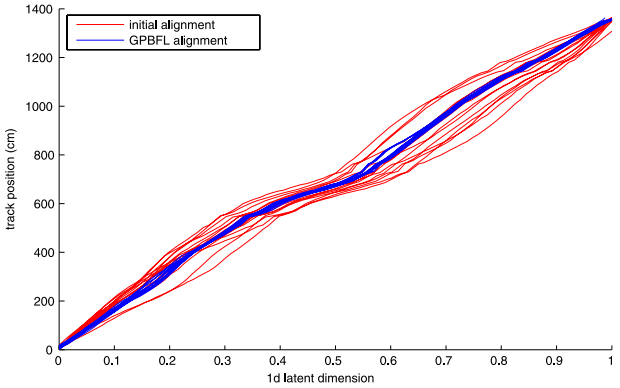

Fig. 10 (Color online) Plot showing alignment of training data in 1D latent space vs. track position. *Red lines* show initial alignment. *Blue* indicates final alignment after GPBF-LEARN optimization. Note that multiple *blue lines* are shown with high overlap, indicating good alignment

5.3.2 Simple trajectory replay

This experiment shows how an imitation control model can be learned based on previous demonstrations. We first show how a naive method for control replay fails. This naive replay method simply plays back the human demonstration based on the timestamps of the controls. The results are shown in Fig. 11. As can be seen, due to system noise, the controls slowly get out of sync with the original demonstration causing a crash around timestep 8,800. This crash occurs based on the mis-alignment between control and track position, resulting in too high control values before a turn. This test shows that simple replay is not robust enough for this system.

We demonstrate the techniques described in Sect. 4.6. At a high level, GP models are learned with GPBF-LEARN to track the robot in a latent space. A mapping between the latent space and the control supplied to the robot can be learned. Finally, to replay the control, the robot is tracked

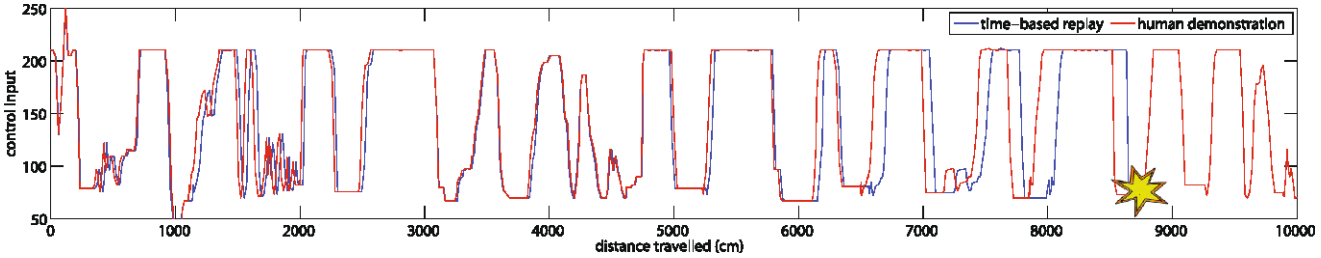

Fig. 11 (Color online) Plot showing time-based replay of human demonstration. The replay controls move out of sync with the original demonstration until the car eventually crashes

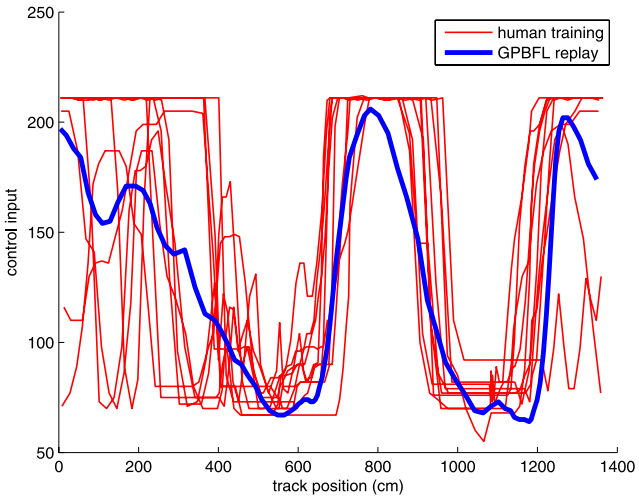

Fig. 12 (Color online) Plot showing typical replay given a number of human demonstrations. The replay controls are much smoother due to the smoothing nature of Gaussian process control model

in the latent space while feeding back the controls learned using the control mapping.

The training data consist of a demonstration of the slot-car by an expert over 16 laps with about 1,400 data points. In order to gain realtime tracking performance the GP-UKF algorithm was reimplemented in C++. The GP-UKF algorithm runs at 40 ± 5 ms on a quad-core dual-processor Intel® Xeon computer running at 3 GHz.

A 3D latent space is used as in the previous experiment from Sect. 5.2. After learning the 3D latent space, we learn a mapping between latent states and controls. This mapping is learned with a Gaussian process with training data $\langle \mathbf{X}, \mathbf{U} \rangle$. Alternatively, a more complex control model could be learned with training data $\langle (\mathbf{X}, \mathbf{X}'), \mathbf{U} \rangle$, but the first formulation gave better performance in tests.

Figure 12 shows the typical control for one lap of the trajectory replay. The controls are plotted against the ground truth track position, and shows both the human demonstrations, and the GPBF-LEARN replay. The replay model gives much smoother control inputs than the demonstrations. This is a result of the smoothing nature of Gaussian process prediction. Essentially, the GP control model blends different demonstration inputs to obtain the final control output. As a result, the trajectory replay is more consistent than the human demonstrations. One way to see this is by the lap times for replay vs human control. The mean laptimes and 95% confidence for human control is 7.10 ± 0.24 sec while for replay it is 7.22 ± 0.17 sec. This shows that the replay gives similar results to human demonstration, but with more consistency (less variance).

In this experiment, the human demonstrator has unique advantages and insight into controlling the system. He can see the track, understand the layout and the dynamics of the car, anticipate turns, etc. The fact that the GPBF-LEARN replay model can control the car as well as the human expert while only having access to very noisy observation data is quite exciting.

5.3.3 *Advanced trajectory replay*

The Barrett WAM™ arm is a 4-DOF highly precise robotic arm with known kinematics. The controls we use for this system are changes in joint angles between time steps. The observations are end effector positions in 3D space

$$\mathbf{U} = [\Delta \mathbf{q}_1, \Delta \mathbf{q}_2, \ldots, \Delta \mathbf{q}_T], \tag{27}$$

$$\mathbf{Z} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_T], \tag{28}$$

where $\mathbf{q}_t$ is the 4D vector of joint angles at time t and $\mathbf{e}_t$ is the 3D vector of the end effector position. We are interested in performing trajectory replay for this system. However, in order to make this an interesting problem, one must assume imprecise control inputs and unknown kinematics. If kinematics are known, then trajectory replay simplifies to the solving for the inverse kinematics of the system. Likewise, if control inputs are highly precise, then trajectory replay can be accomplished with a simple time based replay of the controls.

Figure 13 provides an illustration of the degrees of freedom of the arm. The first DOF is at the base which rotates on the z-axis, the second controls the elevation of the first shaft. The third DOF consists of a twist of that shaft. The last is the "elbow" joint.

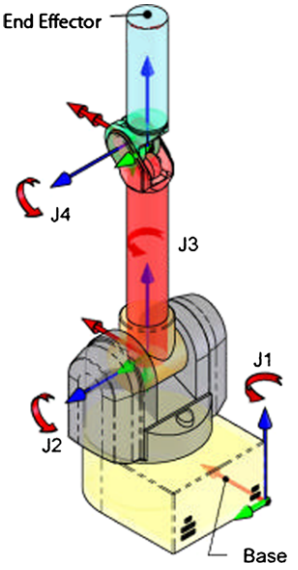

Fig. 13 Illustration showing the degrees of freedom of the robotic arm

We assume that delta joint angles are accurate within 10%. That is,

$$\Delta\hat{\mathbf{q}}_t = \Delta\mathbf{q}_t + \epsilon, \tag{29}$$

$$\epsilon^i \sim \mathcal{N}(0, \tfrac{1}{10}|\Delta\mathbf{q}_t^i|), \tag{30}$$

where $\mathbf{q}_t$ and $\hat{\mathbf{q}}_t$ denotes the recorded and actual joint angles at time t, respectively. The Δ operator describes the change in angles such that $\Delta\mathbf{q}_t = \mathbf{q}_t - \mathbf{q}_{t-1}$. The joint index is i. This setup describes a robotic arm with fairly low cost components which lacks proper encoders and which has unknown or difficult-to-obtain kinematics. The structure of this system is unknown to GPBF-LEARN, which has access only to the recored inputs and observations from the example trajectory. In our test, we generate observations using the forward kinematics of the system. GPBF-LEARN is not given any prior knowledge of this mapping.

These arm experiments focus on the replay aspects of the GPBFL algorithm as described in Sect. 4.6. The purpose of these experiments is to demonstrate trajectory replay in a more complex system than the one presented earlier for the slotcar. The task is to replay a circular trajectory traced out by the arm's end effector in the presence of substantial control noise.

The example trajectory is given by a human demonstrator manually manipulating the WAM™. The trajectory of the end effector from this example is shown in Fig. 14. Overall, 1,000 training points are collected over roughly 50 loops at a sample rate of 10 Hz. Note that the trajectory is not completely circular due to the mechanics of moving the physical arm. Because of the encoder noise, direct replay of the trajectory is not a viable strategy. The end effector trajectory tends to drift as shown in Fig. 15.

We seek to learn a 3D latent representation for this system using GPBF-LEARN. Initialization is performed using

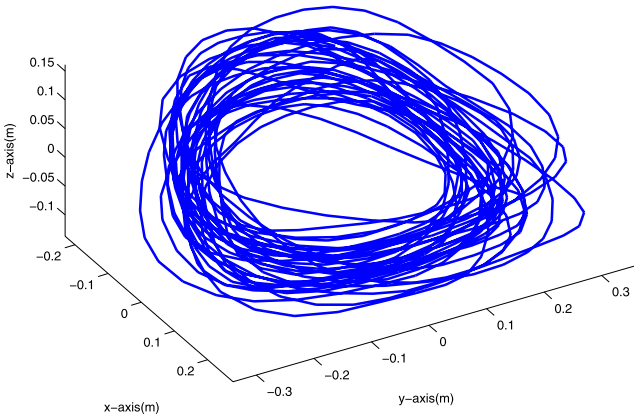

Fig. 14 The end effector position from the demonstration trajectory

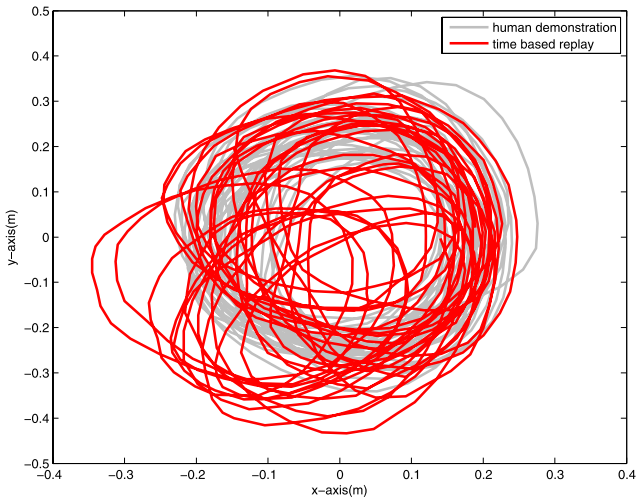

Fig. 15 (Color online) *Top down* view of the end effector trajectory from simple time based replay of the recorded control inputs. The trajectory is affected by noise in the control inputs

N4SID with a 20 step lookahead. This is roughly equivalent to the time it takes to describe a single loop with the end effector. GP-BayesFilter models are built using the optimized latent states. First we test replay using the simple control model where all controls are used as training data as in (26). The control model maps latent states to controls. We then run GP-UKF with an initial pose and control input. The control model predicts controls which are fed back into the system. Observations are generated from the controls given. The system is run for 1,000 timesteps, describing the trajectory shown in Fig. 16. The control model is unable to compensate for the drift, resulting in a failed replay.

We now describe control replay using the advanced control model which uses only a subset of the training data. First, however, we would like to validate the use of GP uncertainty as a measurement of drift. To do this, simulated data of the arm moving in a circular motion is synthesized. This simulated trajectory has much the same shape and ap-

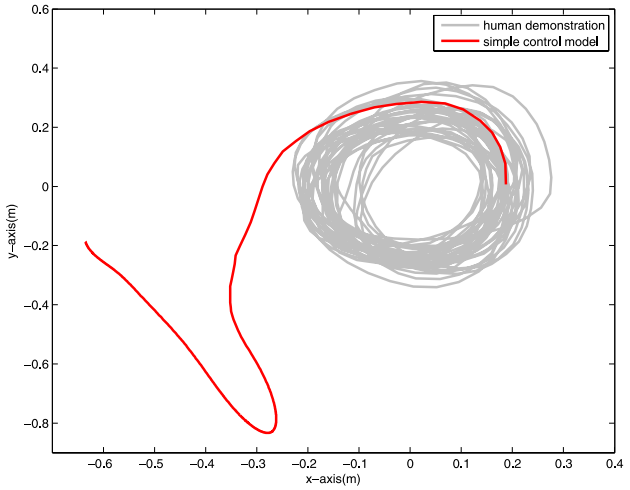

Fig. 16 (Color online) The end effector trajectory using the simple control model results in replay failure

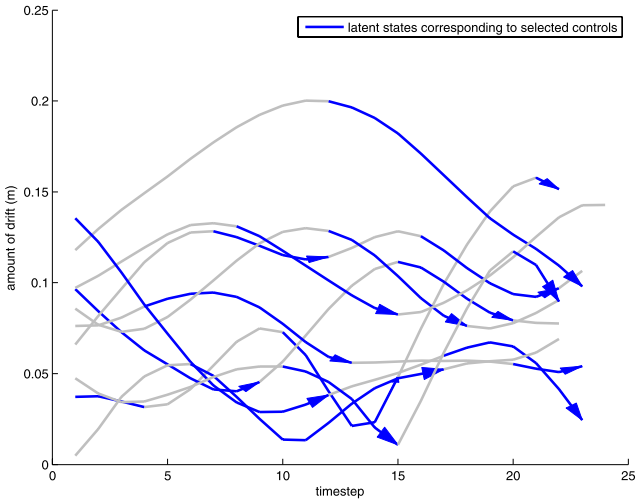

Fig. 17 (Color online) The deviation of the end effector from a true circle for multiple loops. The highlighted segments have corresponding controls which lower GP uncertainty. *The downward trend* indicates the selected controls will tend to reduce drift

pearance as the human demonstration. The only difference is that we know exactly how far the end effector is drifting from the desired true circular trajectory. That is, drift in this case can be directly calculated. Figure 17 shows this deviation of the end effector for 10 loops. The highlighted trajectory segments are ones where the corresponding control lowers GP uncertainty. In general, these are also the controls which push the end effector closer to the circular path. Using these controls will result in a controller which will reduce drift. This trend is less certain for areas near the center of the training manifold (low drift) due to more training data in that area complicating the calculation of uncertainty.

We now evaluate the system with the new control model. The algorithm from Table 2 is used to find the new training data from which we build a new control model. Repeating

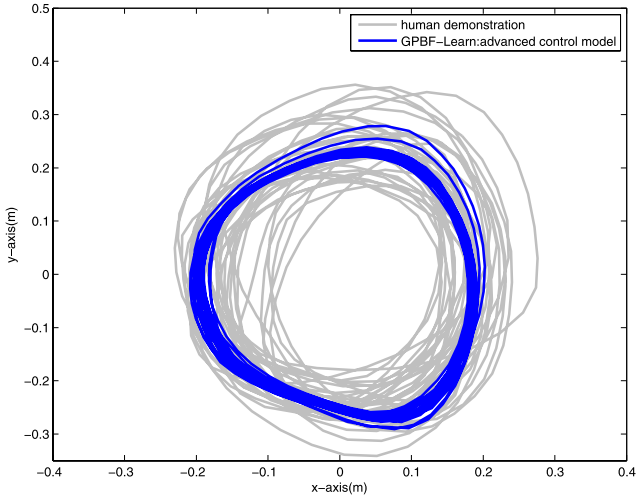

Fig. 18 (Color online) The end effector trajectory using the advanced control model for multiple loops. This model corrects for drift in the control inputs

the previous replay experiment with the new model results in the trajectory shown in Fig. 18. No drift is evident in the replay. Any deviations from the trajectory are caused by the control noise in the system.

These experiments demonstrate a powerful technique for trajectory replay. It does not require extensive knowledge of the system, in particular, no modeling of control inputs is needed. This technique can replay trajectories even in the presence of observational aliasing and control noise.

6 Conclusion

This paper introduced GPBF-LEARN, a framework for learning GP-BayesFilters from only weakly labeled training data. We thereby overcome a key limitation of GP-BayesFilters, which previously required the availability of accurate ground truth states for learning Gaussian process prediction and observation models (Ko and Fox 2008).

GPBF-LEARN builds on recently introduced Gaussian Process Latent Variable Models (GPLVMs) and their extensions to dynamical systems (Lawrence 2005; Wang et al. 2008). GPBF-LEARN improves on existing GPLVM systems in various ways. First, it can incorporate weak labels on the latent states. It is thereby able to learn a latent space that is consistent with a desired physical space, as demonstrated in the context of our slotcar track. Second, GPBF-LEARN can incorporate control information into the dynamics model used for the latent space. Obviously, this ability to use control information is extremely important for complex dynamical systems. Third, we introduce N4SID as a powerful initialization method for GPLVMs. In our slotcar testbed we found that N4SID enabled GPBF-LEARN to learn a model even when the initialization via PCA failed. Our

experiments also show that GPBF-LEARN learns far more consistent models than N4SID alone.

Additional experiments on fully unlabeled data show that GPBF-LEARN can perform nonlinear system identification and data alignment. We demonstrate this ability in the context of tracking a slotcar solely based on control and IMU information. Here, our approach is able to learn a consistent 3D latent space solely based on the control and observation sequence. This application is challenging, since the observations are not very informative and show a high rate of aliasing. Furthermore, due to the constraints of the track, the dynamics and observation model of the car strongly depend on the layout of the track. Thus, GPBF-LEARN has to jointly recover a model for the car and the track. Additionally, GPBF-LEARN is shown to compare favorable to a state-of-the-art subspace identification algorithm on a sample non-linear system.

Finally, our experiments has shown some unique aspects of GPBF-LEARN by showing time alignment in 1D latent space, and also demonstration replay on two robotics systems. Trajectory replay using GPBF-LEARN, in particular, is a powerful technique that requires little prior knowledge of the system and operates with dynamics and observation noise.

Possible extensions of this work include the incorporation of parametric models to improve learning and generalization. Finally, the latent model underlying GPBF-LEARN is by no means restricted to GP-BayesFilters. It can be applied to improve learning quality whenever there is no accurate ground truth data available for training Gaussian processes.

Acknowledgements We would like to thank Michael Chung and Deepak Verma for their help in running the slotcar experiments. We want to thank Louis LeGrand for his support of the Intel InertiaDot IMU. Acknowledgment goes out to Yoshinobu Kawahara for providing sample code for KCCA subspace identification method. This work was supported in part by ONR MURI grants number N00014-07-1-0749 and N00014-09-1-1052, and by the NSF under contract numbers IIS-0812671 and BCS-0508002.

Appendix: N4SID

N4SID is a system identification algorithm that was developed Peter Van Overschee and Bart De Moor in 1994. It can be used to identify the parameters and latent state of a linear dynamical system assuming the following system model:

$$\mathbf{x_{t+1}} = A\mathbf{x_t} + B\mathbf{u_t} + \rho_x, \tag{31}$$

$$\mathbf{y_t} = C\mathbf{x_t} + \rho_y, \tag{32}$$

where ρ_x and ρ_y represent Gaussian noise.

Table 5 The N4SID algorithm

Algorithm N4SID (Y, U, n, i)**:**

1: *Construct Hankel matrices*
2: *Perform oblique projection*
3: *Eigendecompose projection*
4: *Recover latent states X*
5: *Find least squared solution for system matrices*
6: *Calculate covariance matrices*
7: *return system matrices A,B,C, covariance matrices R,Q, and latent states X*

The latent states are derived via operations on the block Hankel matrices of the future outputs, future inputs, and past input/outputs, respectively:

$$Y_f = \begin{bmatrix} y_{i+1} & y_{i+2} & \cdots & y_{i+j} \\ \cdots & \cdots & \cdots & \cdots \\ y_{2i-1} & y_{2i} & \cdots & y_{2i+j-2} \end{bmatrix}, \tag{33}$$

$$U_f = \begin{bmatrix} u_{i+1} & u_{i+2} & \cdots & u_{i+j} \\ \cdots & \cdots & \cdots & \cdots \\ u_{2i-1} & u_{2i} & \cdots & u_{2i+j-2} \end{bmatrix}, \tag{34}$$

$$W_p = \begin{bmatrix} u_0 & u_1 & \cdots & u_{j-1} \\ \vdots & \vdots & \vdots & \vdots \\ u_{i-1} & u_i & \cdots & u_{i+j-2} \\ y_0 & y_1 & \cdots & y_{j-1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{i-1} & y_i & \cdots & y_{i+j-2} \end{bmatrix}, \tag{35}$$

where i is the number of timesteps into the past and future considered by the algorithm and j is the total number of timesteps minus $2i$.

At a very high level, the intuition behind this algorithm is to project the future outputs Y_f *along* future inputs U_f and onto the past inputs and outputs W_p (this is called an oblique projection (Meyer 2000)). This projection finds the best prediction of future outputs based on inputs and past outputs. A set of approximate latent states is then found by a spectral decomposition of the resulting projection. Finally the parameters of the linear dynamical system are recovered by regression using the latent states and Hankel matrices.

The steps of the N4SID algorithm are detailed in Table 5, where n is the desired dimensionality of the latent states. The system parameters returned by N4SID are unbiased and can be used for state estimation within the context of a linear Kalman filter.

References

Abbeel, P., Dolgov, D., Ng, A., & Thrun, S. (2008). Apprenticeship learning for motion planning with application to parking lot nav-

igation. In *Proc. of the IEEE/RSJ international conference on intelligent robots and systems, IROS*.

Boots, B., Siddiqi, S., & Gordon, G. (2009). Closing the learning-planning loop with predictive state representations. http://arxiv.org/abs/0912.2385

Bowling, M., Wilkinson, D., Ghodsi, A., & Milstein, A. (2005). Subjective localization with action respecting embedding. In *Proc. of the international symposium of robotics research, ISRR*.

Coates, A., Abbeel, P., & Ng, A. (2008). Learning for control from multiple demonstrations. In *Proc. of the international conference on machine learning, ICML*.

Deisenroth, M., Huber, M., & Hanebeck, U. (2009). Analytic moment-based Gaussian process filtering. In *Proc. of the international conference on machine learning, ICML* (pp. 225–232). New York: ACM.

Ekvall, S., & Kragic, D. (2004). Interactive grasp learning based on human demonstration. In *Proc. of the IEEE international conference on robotics & automation, ICRA* (pp. 3519–3524).

Engel, Y., Szabo, P., & Volkinshtein, D. (2006). Learning to control an octopus arm with Gaussian process temporal difference methods. In *Advances in neural information processing systems, NIPS* (Vol. 18).

Ferris, B., Hähnel, D., & Fox, D. (2006). Gaussian processes for signal strength-based location estimation. In *Proc. of robotics: science and systems, RSS*.

Ferris, B., Fox, D., & Lawrence, N. (2007). WiFi-SLAM using Gaussian process latent variable models. In *Proc. of the international joint conference on artificial intelligence, IJCAI*.

Grimes, D., & Rao, R. (2008). Learning nonparametric policies by imitation. In *Proc. of the IEEE/RSJ international conference on intelligent robots and systems, IROS* (pp. 2022–2028).

Hsu, E., Pulli, K., & Popović, J. (2005). Style translation for human motion. *ACM Transactions on Graphics*, *24*, 1082–1089.

Hsu, E., da Silva, M., & Popovic, J. (2007). Guided time warping for motion editing. In *Symposium on computer animation '07 proceedings* (pp. 45–52). Aire-la-Ville: Eurographics Association.

Kawahara, Y., Yairi, T., & Machida, K. (2007). A kernel subspace method by stochastic realization for learning nonlinear dynamical systems. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems* (Vol. 19, pp. 665–672). Cambridge: MIT Press.

Kersting, K., Plagemann, C., Pfaff, P., & Burgard, W. (2007). Most likely heteroscedastic Gaussian process regression. In *Proc. of the international conference on machine learning, ICML*.

Ko, J., & Fox, D. (2008). GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. In *Proc. of the IEEE/RSJ international conference on intelligent robots and systems, IROS*.

Ko, J., & Fox, D. (2009). Learning GP-Bayesfilters via Gaussian process latent variable models. In *Proc. of robotics: science and systems, RSS*.

Ko, J., Klein, D., Fox, D., & Hähnel, D. (2007). Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Proc. of the IEEE international conference on robotics & automation, ICRA*.

Ko, J., Klein, D., Fox, D., & Hähnel, D. (2007). GP-UKF: Unscented Kalman filters with Gaussian process prediction and observation models. In *Proc. of the IEEE/RSJ international conference on intelligent robots and systems, IROS*.

Lawrence, N. (2003). Gaussian process latent variable models for visualization of high dimensional data. In *Advances in neural information processing systems, NIPS*.

Lawrence, N. (2005). Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, *6*, 1783–1816.

Lawrence, N., & Moore, A. J. (2007). Hierarchical Gaussian process latent variable models. In *Proc. of the international conference on machine learning, ICML*.

Lawrence, N., & Quiñonero Candela, J. (2006). Local distance preservation in the GP-LVM through back constraints. In *Proc. of the international conference on machine learning, ICML*.

Littman, M., Sutton, R., & Singh, S. (2001). Predictive representations of state. In *Advances in neural information processing systems, NIPS* (Vol. 14, pp. 1555–1561). Cambridge: MIT Press.

Ljung, L. (1987). *System identification*. New York: Prentice Hall.

Meyer, C. D. (Ed.) (2000). *Matrix analysis and applied linear algebra*. Philadelphia: Society for Industrial and Applied Mathematics.

Nguyen-Tuong, D., Seeger, M., & Peters, J. (2008). Local Gaussian process regression for real time online model learning and control. In *Advances in neural information processing systems, NIPS* (Vol. 22).

Plagemann, C., Fox, D., & Burgard, W. (2007). Efficient failure detection on mobile robots using Gaussian process proposals. In *Proc. of the international joint conference on artificial intelligence, IJCAI*.

Rabiner, L., Rosenberg, A., & Levinson, S. (1978). Considerations in dynamic time warping algorithms for discrete word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *26*(6), 575–582.

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. In *Advances in neural information processing systems, NIPS*.

Rasmussen, C. E., & Williams, C. K. I. (2005). *Gaussian processes for machine learning*. Cambridge: MIT Press.

Schmill, M., Oates, T., & Cohen, P. (1999). Learned models for continuous planning. In *Proceedings of uncertainty 99: the 7th international workshop on artificial intelligence and statistics* (pp. 278–282). Los Altos: Kaufmann.

Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Deylon, B., Glorennec, P., Hjalmarsson, H., & Juditsky, A. (1995). Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, *31*, 1691–1724.

Snelson, E., & Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In *Advances in neural information processing systems, NIPS* (Vol. 18).

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. Cambridge: MIT Press. ISBN 0-262-20162-3.

Urtasun, R., Fleet, D., & Fua, P. (2006). Gaussian process dynamical models for 3D people tracking. In *Proc. of the IEEE computer society conference on computer vision and pattern recognition, CVPR*.

Van Overschee, P., & De Moor, B. (1996). *Subspace identification for linear systems: theory, implementation, applications*. Norwell: Kluwer Academic.

Verdult, V., Suykens, J., Boets, J., Goethals, I., De Moor, B., & Leuven, K. (2004). Least squares support vector machines for kernel in nonlinear state-space identification. In *Proceedings of the 16th international symposium on mathematical theory of networks and systems, MTNS2004*.

Wang, J., Fleet, D., & Hertzmann, A. (2008). Gaussian process dynamical models for human motion. In *IEEE transactions on pattern analysis and machine intelligence, PAMI*.

Zhou, F., & De la Torre, F. (2009). Canonical time warping for alignment of human behavior. In *Advances in neural information processing systems, NIPS*.

Jonathan Ko is a graduate student at the Department of Computer Science at the University of Washington, Seattle, as a student of Prof. Dieter Fox. His main interest is in machine learning for robotic systems, in particular, Gaussian processes for modeling of dynamical systems. He won the Sarcos Best Student Paper Award in IROS 2007 for his paper titled "GP-UKF: Unscented Kalman Filters with Gaussian Process Prediction and Observation Models".

Dieter Fox is Associate Professor in the Computer Science & Engineering Department at the University of Washington, Seattle, and Director of the Intel Labs Seattle. He obtained his Ph.D. from the University of Bonn, Germany. Before joining UW, he spent two years as a postdoctoral researcher at the CMU Robot Learning Lab. His research focuses on probabilistic state estimation in robotics and activity recognition. Dr. Fox has published over 100 technical papers and is co-author of the text book "Probabilistic Robotics". He was program co-chair of the Twenty-Third Conference on Artificial Intelligence (AAAI-08) and has been on the editorial board of the Journal of Artificial Intelligence Research and the IEEE Transactions on Robotics. He has received several awards for his research, including an NSF CAREER award and best paper awards at robotics and Artificial Intelligence conferences.