Learning GP-BayesFilters via Gaussian Process Latent Variable Models

Jonathan Ko **Dieter Fox**
University of Washington, Department of Computer Science & Engineering, Seattle, WA

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 learn non-parametric filter 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 prohibitive overhead. In this paper we introduce GPBF-LEARN, a framework for training GP-BayesFilters without any 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 IMU measurements only.**

I. INTRODUCTION

Over the last years, Gaussian processes (GPs) have been applied with great success to robotics tasks such as reinforcement learning [3] and learning of prediction and observation models [5, 16, 8]. GPs learn probabilistic regression models from training data consisting of input-output examples [17]. 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 filtering techniques, most easily into particle filters [5, 16].

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 [9, 7]. 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 [7]. 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 [8]. This requirement limits the 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 [17]. To overcome this limitation, Lawrence [11] recently introduced Gaussian Process Latent Variable Models (GPLVM) for probabilistic, non-linear principal component analysis. 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, GPLVMs learn an embedding of the output examples into 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 [4, 21, 19, 12].

In this paper we introduce GPBF-LEARN, a framework for learning GP-BayesFilters from partially or fully unlabeled training data. The input 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 [7]. 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.

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 turn information. Using this testbed, we demonstrate that GPBF-LEARN outperforms alternative approaches to learning GP-BayesFilters. We furthermore show that GPBF-LEARN can be used to automatically align multiple demonstration traces and learn a filter from completely unlabeled data.

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 Section IV, we introduce the GPBF-LEARN framework. Experimental results are given in Section V, followed by a discussion.

II. RELATED WORK

Lawrence [11] introduced Gaussian Process Latent Variable Models (GPLVMs) for visualization of high-dimensional data. 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 [13], 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 [4].

Wang and colleagues [21] 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 [21] and visual tracking [19] 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 [19] is not designed for real-time or near real-time performance, nor does is provide uncertainty estimates as GP-BayesFilters. Other alternatives for non-linear embedding in the context of dynamical systems are hierarchical GPLVMs [12] and action respecting embeddings (ARE) [1]. 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 [14, 20]. The goal of these techniques is the same as that of GPBF-LEARN, namely to learn a model for a dynamical system from sequences of control inputs and observations. The model underlying N4SID [20] is a linear Kalman filter. Due to its flexibility and robustness, N4SID is extremely popular. It has been applied successfully for human motion animation [6]. In our experiments, we demonstrate that GPBF-LEARN provides superior performance due to its ability to model non-linear systems. We also show that N4SID provides excellent initialization for GPLVMs for dynamical systems.

III. PRELIMINARIES

This section provides background on Gaussian Processes (GPs) 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.

A. Gaussian Process Regression

Gaussian processes (GP) are non-parametric techniques for learning regression functions from sample data [17]. Assume we have n d-dimensional input vectors: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]$. A GP defines a zero-mean, Gaussian prior distribution over the outputs $\mathbf{y} = [y_1, y_2, ..., y_n]$ at these values [1]:

$$p(\mathbf{y} \mid \mathbf{X}) = \mathcal{N}(\mathbf{y};\, 0,\, \mathbf{K}_y + \sigma_n^2 \mathbf{I}), \tag{1}$$

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 distance measure, we see that if points $\mathbf{x}_i$ and $\mathbf{x}_j$ are close in the input 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:

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

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. σ_f^2 is the signal variance.

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 (1) on the training data and $\mathbf{x}_*$ results in a Gaussian predictive distribution over the corresponding output y_*

$$p(y_* \mid \mathbf{x}_*, D) = \mathcal{N}\left(y_*, \mathrm{GP}_\mu\left(\mathbf{x}_*, D\right), \mathrm{GP}_\Sigma\left(\mathbf{x}_*, D\right)\right) \tag{3}$$

with mean

$$\mathrm{GP}_\mu\left(\mathbf{x}_*, D\right) = \mathbf{k}_*^T [K + \sigma_n^2 I]^{-1} \mathbf{y} \tag{4}$$

and variance

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

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 [17]. Making the dependency on hyperparameters explicit, we get

$$\boldsymbol{\theta}_y^* = \underset{\boldsymbol{\theta}_y}{\operatorname{argmax}}\ \log\ p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}_y). \tag{6}$$

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 $\mathbf{y}$.

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

B. Gaussian Process Latent Variable Models

GPLVMs were introduced in the context of visualization of high-dimensional data [10]. GPLVMs perform nonlinear dimensionality reduction in the context of Gaussian processes. The underlying probabilistic model is still a GP regression model as defined in (1). 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:

$$\langle \mathbf{X}^*, \boldsymbol{\theta}_y^* \rangle = \underset{\mathbf{X}, \boldsymbol{\theta}_y}{\operatorname{argmax}} \log \ p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}_y) \tag{7}$$

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 [11, 21].

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 [21] and hierarchical GPLVMs [12], 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:

$$p(\mathbf{Y}, \mathbf{X}, \boldsymbol{\theta}_y, \boldsymbol{\theta}_x) = p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}_y) p(\mathbf{X} \mid \boldsymbol{\theta}_x) p(\boldsymbol{\theta}_y) p(\boldsymbol{\theta}_x) \tag{8}$$

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

$$p(\mathbf{X} \mid \boldsymbol{\theta}_x) = \mathcal{N}(\mathbf{X}; 0, \mathbf{K}_x + \sigma_m^2 \mathbf{I}), \tag{9}$$

where $\mathbf{K}_x$ is an appropriate kernel matrix. In Section IV, 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 (8):

$$\langle \mathbf{X}^*, \boldsymbol{\theta}_y^*, \boldsymbol{\theta}_x^* \rangle = \underset{\mathbf{X}, \boldsymbol{\theta}_y, \boldsymbol{\theta}_x}{\operatorname{argmax}} \Big(\log p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}_y) + \log p(\mathbf{X} \mid \boldsymbol{\theta}_x) + \log p(\boldsymbol{\theta}_y) + \log p(\boldsymbol{\theta}_x) \Big) \tag{10}$$

Such extensions to GPLVMs have been used successfully to model temporal data such as motion capture sequences [21, 12] and visual tracking data [19].

C. 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, $p(\mathbf{z}_t \mid \mathbf{x}_t)$. The prediction model describes how the state $\mathbf{x}_t$ changes based on time and control input $\mathbf{u}_{t-1}$, and the observation model describes the likelihood of making an observation $\mathbf{z}_t$ given the state $\mathbf{x}_t$. In robotics, these models are typically parametric descriptions of the underlying processes, see [18] 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 [7]. 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

$$D_p = \langle (\mathbf{X}, \mathbf{U}), \mathbf{X}' \rangle$$
$$D_o = \langle \mathbf{X}, \mathbf{Z} \rangle ,$$

where $\mathbf{X}$ is a matrix containing the sequence of ground truth states, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T]$, $\mathbf{X}'$ is a matrix containing the state changes, $\mathbf{X}' = [\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_2, \ldots, \mathbf{x}_T - \mathbf{x}_{T-1}]$, and $\mathbf{U}$ and $\mathbf{Z}$ contain the sequences of controls and observations, respectively. By plugging these training sets into (4) and (5), one gets GP prediction and observation models mapping from a state, $\mathbf{x}_{t-1}$, and a control, $\mathbf{u}_{t-1}$, to change in state, $\mathbf{x}_t - \mathbf{x}_{t-1}$, and from a state, $\mathbf{x}_t$, to an observation, $\mathbf{z}_t$, respectively. These probabilistic models can be readily incorporated into Bayes filters such as particle filters and unscented Kalman filters. An additional derivative of (4) provides the Taylor expansion needed for extended Kalman filters [7].

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 [7, 15, 16] or manual labeling [5], 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.

IV. GPBF-Learn

In this section we show how GP-BayesFilters can be learned from weakly labeled data. While the extensions of GPLVMs described in Section III-B 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:

$$p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid \mathbf{Z}, \mathbf{U}, \widehat{\mathbf{X}}) \quad \propto \quad p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z)\, p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x)\, p(\mathbf{X} \mid \widehat{\mathbf{X}})\, p(\boldsymbol{\theta}_z) p(\boldsymbol{\theta}_x) \quad (11)$$

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 (11) is the same as in the GPLVM for dynamical systems (8), 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.

A. 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:

$$p(\mathbf{X} \mid \widehat{\mathbf{X}}) \;=\; \prod_{\hat{\mathbf{x}}_t \in \widehat{\mathbf{X}}} \mathcal{N}(\mathbf{x}_t; \hat{\mathbf{x}}_t, \sigma^2_{\hat{\mathbf{x}}_t}) \quad (12)$$

Here, $\sigma^2_{\hat{\mathbf{x}}_t}$ is the uncertainty in label $\hat{\mathbf{x}}_t$. As note 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.

B. 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 (11), 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

$$p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x) = \mathcal{N}(\mathbf{X}^{\text{out}}; 0, \mathbf{K}_{\text{x}} + \sigma_x^2 \mathbf{I}) \;, \quad (13)$$

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: $\mathbf{K}_{\text{x}}[t, t'] = k\left(\mathbf{x}_t^{\text{in}}, \mathbf{x}_{t'}^{\text{in}}\right)$, where $\mathbf{x}_t^{\text{in}}$ and $\mathbf{x}_{t'}^{\text{in}}$ 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. To see, 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:

$$\mathbf{x}_t^{\text{in}} \;=\; \mathbf{x}_{t-1} \quad (14)$$
$$\mathbf{x}_t^{\text{out}} \;=\; \mathbf{x}_t \quad (15)$$

Optimization with such a dynamics model encourages smooth state sequences $\mathbf{X}$. Generating smooth *velocities* can be achieved by setting $\mathbf{x}_t^{\text{in}}$ to $\dot{\mathbf{x}}_{t-1}$ and $\mathbf{x}_t^{\text{out}}$ to $\dot{\mathbf{x}}_t$, where $\dot{\mathbf{x}}_t$ represents the velocity $[\mathbf{x}_t - \mathbf{x}_{t-1}]$ at time t [21]. 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:

$$\mathbf{x}_t^{\text{in}} \;=\; [\mathbf{x}_{t-1}, \dot{\mathbf{x}}_{t-1}, \mathbf{u}_{t-1}]^T \quad (16)$$
$$\mathbf{x}_t^{\text{out}} \;=\; \dot{\mathbf{x}}_t \quad (17)$$

This model encourages smooth changes in velocity depending on control input. By adding $\mathbf{x}_{t-1}$ to $\mathbf{x}_t^{\text{in}}$, 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.

C. Optimization

Just as regular GPLVM models, GPBF-LEARN determines the unknown values of the latent states $\mathbf{X}$ by optimizing the log of posterior over the latent state sequence and the hyperparameters. The log of (11) is given by

$$\log p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid D) = \log p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z) + \log p(\mathbf{X} \mid \mathbf{U}, \boldsymbol{\theta}_x) + \log p(\mathbf{X} \mid \widehat{\mathbf{X}}) + \log p(\boldsymbol{\theta}_z) + \log p(\boldsymbol{\theta}_x) + \text{const} \;, \quad (18)$$

where D represents the training data $[\mathbf{Z}, \mathbf{U}, \widehat{\mathbf{X}}]$. We perform this optimization using scaled conjugate gradient descent [21]. The gradients of the log are given by:

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

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

$$\frac{\partial \log p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid D)}{\partial \boldsymbol{\theta}_z} = \frac{\partial \log p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}_z)}{\partial \boldsymbol{\theta}_z} + \frac{\partial \log p(\boldsymbol{\theta}_z)}{\partial \boldsymbol{\theta}_z} \;. \quad (21)$$

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

1:  if ($\widehat{\mathbf{X}} \neq \emptyset$)
        $\mathbf{X}$                  $:= \widehat{\mathbf{X}}$
    else
        $\mathbf{X}$                  $:= N4SID_x(\mathbf{Z}, \mathbf{U})$
2:  $\langle \mathbf{X}^*, \boldsymbol{\theta}_x^*, \boldsymbol{\theta}_z^* \rangle$   $:=$ SCG_optimize $\left(\log p(\mathbf{X}, \boldsymbol{\theta}_x, \boldsymbol{\theta}_z \mid \mathbf{Z}, \mathbf{U}, \widehat{\mathbf{X}})\right)$
3:  $\mathbf{GPBF}$               $:=$ Learn_gpbf$(\mathbf{X}^*, \mathbf{U}, \mathbf{Z})$
4:  return $\mathbf{GPBF}$
```

TABLE I
THE GPBF-LEARN ALGORITHM.

The individual derivatives follow as

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

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 Gram matrix with respect to $\mathbf{X}$ or the hyperparameters, respectively.

D. GPBF-LEARN *Algorithm*

A high level overview of the GPBF-LEARN algorithm is given in Table I. 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 [20]. In our experiments, N4SID provides initialization that is far superior to the standard PCA initialization used by [11, 21]. Then, in Step 2, scaled conjugate gradient (SCG) descent determines the latent states and hyperparameters via optimization of the log posterior (18). This iterative procedure computes the gradients (19) – (21) 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 Section III-C.

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.

V. EXPERIMENTS

In these experiments we evaluate different properties of GPBF-LEARN using the computer controlled slotcar platform shown in Fig. 1. Specifically, we demonstrate the ability of GPBF-LEARN to incorporate prior knowledge over the latent states, to learn robust GP-BayesFilters from noisy and sparse labeled data, and to perform system identification without any ground truth states.

In an additional experiment not reported here, we compared the two dynamics models described in Section IV-B. Using 10-step ahead prediction as evaluation criteria, we found that our control based model (17) significantly outperforms the simpler model (15) that is typically used for GPLVMs. In fact, our model reduces the prediction error by almost 50%, from 29.2 to 16.1 cm.

A. Slotcar evaluation platform

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 14m. An overhead camera tracks the car and is used as ground truth data for evaluation of the algorithms. The car is a standard 1:32 scale model manufactured by Carrera International and augmented with a Microstrain 3DM-GX1 inertial measurement unit (IMU), as shown in the next panel in Fig. 1. The IMU tracks the relative orientation of the car. These measurements are sent off-board in real-time via a WiFi interface. Control signals to the car are supplied by an offboard computer. These controls signals are directly proportional to the amperage supplied the the car motor.

The data, $\langle \mathbf{Z}, \mathbf{U} \rangle$, is collected at 15 frames per second. From the IMU data, we extract the 3D orientation of the car in Euler angles. During the evaluations, we take the *difference* between consecutive angle readings as the observation data $\mathbf{Z}$. The raw angles could potentially be used as observations as well. However, even though this might make tracking and system identification easier over short periods of time, we observed substantial IMU angle drift and thus decided to use a more realistic scenario that does not depend on a global heading sensor. As can be seen in the third panel in Fig. 1, the resulting turning rate data is very noisy and includes substantial amounts of aliasing, in which the same angle 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. This kind of aliasing makes learning the latent space particularly challenging since it does not provide a unique mapping from the observation sequence $\mathbf{Z}$ to the latent space $\mathbf{X}$.

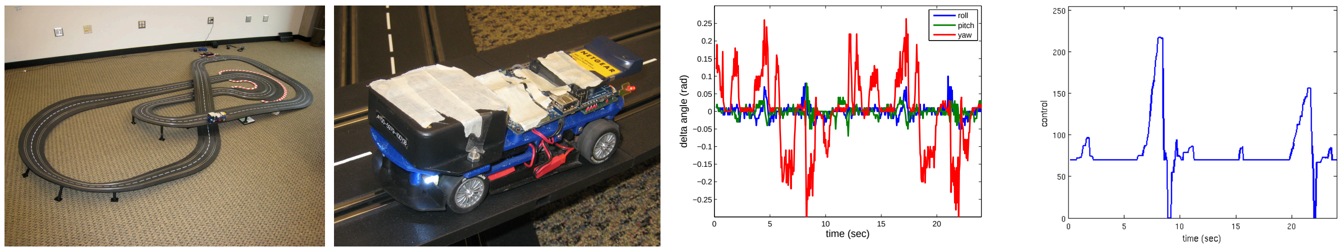

Fig. 1. (left) The slotcar track used during the experiments. An overhead camera is used to ground truth evaluation. (left middle) The test vehicle moves along a slot in the track, velocity control is provided remotely by a desktop PC. The state of the vehicle is estimated based on an on-board IMU. (right middle) IMU turning rate in roll, pitch, and yaw. Shown is data collected over two rounds around the track. (right) Control inputs for the same run.

In all experiments we use a GP-UKF to generate tracking results [9]. In the first set of experiments 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 $\widehat{\mathbf{X}}$, as below.

B. *Incorporating noisy labels*

Here 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 Section IV-A. We call this approach Gpdm, since it results from applying the model of Wang *et.al.* [21] 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 3D IMU observations $\mathbf{Z}$ along with the control information $\mathbf{U}$ into a 1D latent space $\mathbf{X}$. Training data consist 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 via background subtraction and simple blob tracking, followed by snapping xy pixel locations to an aligned model of the track. 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. Currently, learning is done with each loop treated as a separate episode as we do not handle the jump between the beginning and end of the loop for the 1D latent space. This could likely be handled by a periodic kernel as future work. 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 left panel in Fig. 2 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. 2 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.

C. *Incorporating sparse labels*

In some settings it might not be possible to provide even noisy labels for all training points. Here we evaluate 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 right panel in Fig. 2 shows the errors between ground truth 1D latent space and the learned latent space, $\mathbf{X}^*$, for different

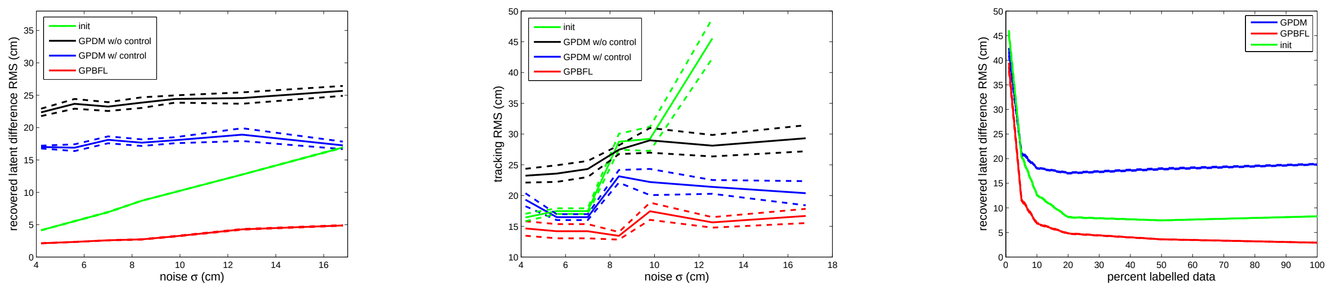

Fig. 2. Evaluation of INIT, GPDM, and GPBFL on noisy and sparse labels. Dashed lines provide 95% confidence intervals. (left) Difference between the learned latent states $\mathbf{X}^*$ and ground truth as a function of noise level in the labels $\widehat{\mathbf{X}}$. (middle) Tracking errors for different noise levels. (right) Difference between the learned latent states and ground truth as a function of label sparsity.

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 it does not learn anything and thereby avoids drifting from the provided labels.

D. GPBF-LEARN *for subspace identification without labels*

The final experiment demonstrates that GPBF-LEARN can learn a model without any labeled data. Here, the training input consists solely of turning rate observations $\mathbf{Z}$ and control inputs $\mathbf{U}$. No weak labels $\widehat{\mathbf{X}}$ are provided and no information about the structure of the latent space is given. To encode less knowledge about the underlying race track, we make GPBF-LEARN learn a 2D latent space. 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 [21, 11, 19]. In this case, GPBF-LEARN was not able to learn a smooth model of the latent space. This is because PCA does not take the dynamics in latent space into account.

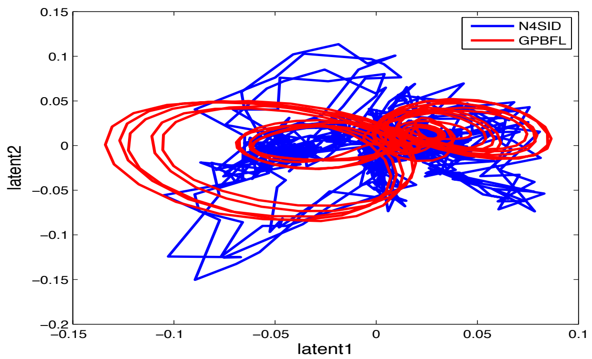

Fig. 3. 2D latent space learned by N4SID and GPBF-LEARN with N4SID initialization.

A different approach for initialization is N4SID, which is a well known, linear model for system identification of dynamical systems [20]. N4SID provides an estimate of the hidden state which does take into account the system dynamics. The latent space recovered by N4SID is given by the blue graph in Fig. 3. N4SID can only generate an extremely un-smooth latent space that does not reflect the smooth structure of the underlying track. When running GPBF-LEARN on the data, initialized with N4SID, we get the red graph shown in the same figure. Obviously, GPBF-LEARN takes advantage of its underlying non-linear GP model to recover a smooth latent space that nicely reflects the cyclic structure of the race track. Note that we would not expect all cycles through the track to be mapped exactly on top of each other, since the slotcar has very different observations depending on its velocity.

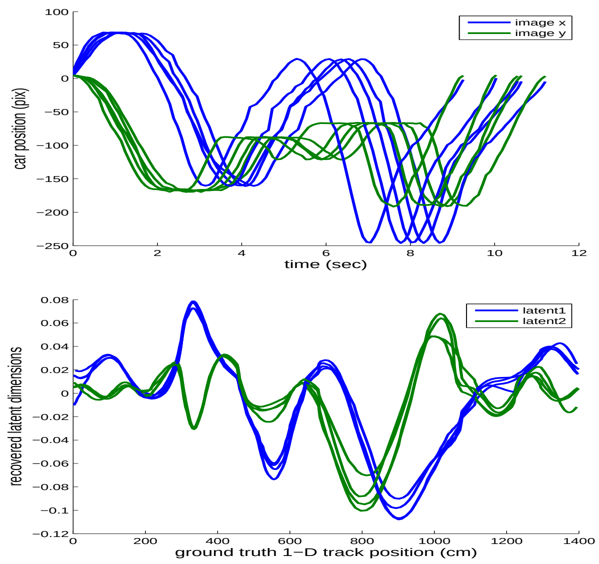

Fig. 4. Plots showing misalignment of the input data (top), and alignment in the latent space learned by GPBF-LEARN (bottom).

An important aspect of the optimization problem is to learn a proper alignment of the data, that is, each portion of the real track should correspond to similar latent states. The top of Fig. 4 shows the true x and y positions of the car in physical space for the different cycles through the track. As can be seen, the car moves through the track with different velocities, resulting in the mis-aligned graphs. To visualize that GPBF-LEARN was able to recover an aligned latent space from the unaligned input data (note that we used IMU, xy is only used for visualization), we plot the two latent dimensions vs. the position on the 1D track model. This plot is shown in the bottom of Fig. 4. As can be seen, the latent states are well aligned with the 1D model over the different cycles through the track. This result is extremely encouraging, since it shows that we might be able to learn an imitation control model based on such demonstrations, as done by Coates and colleagues [2].

VI. 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 so far required the availability of accurate ground truth states for learning Gaussian process prediction and observation models [7].

GPBF-Learn builds on recently introduced Gaussian Process Latent Variable Models (GPLVMs) and their extensions to dynamical systems [11, 21]. 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, a linear subspace ID technique, as a very 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 subspace identification and data alignment. We demonstrate this ability in the context of tracking a slotcar on a track solely based on control and IMU turn rate information. Here, our approach is able to learn a consistent 2D latent space solely based on the control and observation sequence. This application is extremely challenging, since the observations are not very informative and show a high rate of aliasing. Furthermore, due to the constraint onto 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.

We have also obtained some preliminary results of tracking the slotcar in the 3D latent space. For this task, the automatically learned GP hyperparameters turned out to be insufficient for tracking, requiring additional manual tuning. To overcome this problem, we intend to explore the use of discriminative learning to optimize the hyperparameters for filtering.

In future work, GPBF-Learn could be applied to imitation learning, similar to the approach introduced by Coates and colleagues for helicopter control [2]. In this context we would take advantage of the automatic alignment of different demonstrations given by GPBF-Learn . An integration of GP-BayesFilter with model predictive control techniques is an interesting question in this context. Other possible extensions 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.

Acknowledgments

We would like to thank Michael Chung and Deepak Verma for their help in running the slotcar experiments. This work was supported in part by ONR MURI grant number N00014-07-1-0749 and by the NSF under contract numbers IIS-0812671 and BCS-0508002.

References

[1] M. Bowling, D. Wilkinson, A. Ghodsi, and A. Milstein. Subjective localization with action respecting embedding. In *Proc. of the International Symposium of Robotics Research (ISRR)*, 2005.

[2] A. Coates, P. Abbeel, and A. Ng. Learning for control from multiple demonstrations. In *Proc. of the International Conference on Machine Learning (ICML)*, 2008.

[3] Y. Engel, P. Szabo, and D. Volkinshtein. Learning to control an octopus arm with Gaussian process temporal difference methods. In *Advances in Neural Information Processing Systems 18 (NIPS)*, 2006.

[4] B. Ferris, D. Fox, and N. Lawrence. WiFi-SLAM using Gaussian process latent variable models. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[5] B. Ferris, D. Hähnel, and D. Fox. Gaussian processes for signal strength-based location estimation. In *Proc. of Robotics: Science and Systems (RSS)*, 2006.

[6] E. Hsu, K. Pulli, and J. Popović. Style translation for human motion. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 2005.

[7] J. Ko and D. Fox. 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)*, 2008.

[8] J. Ko, D. Klein, D. Fox, and D. Hähnel. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2007.

[9] J. Ko, D. Klein, D. Fox, and D. Hähnel. 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)*, 2007.

[10] N. Lawrence. Gaussian process latent variable models for visualization of high dimensional data. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[11] N. Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research (JMLR)*, 6, 2005.

[12] N. Lawrence and A. J. Moore. Hierarchical gaussian process latent variable models. In *Proc. of the International Conference on Machine Learning (ICML)*, 2007.

[13] N. Lawrence and J. Quiñonero Candela. Local distance preservation in the gp-lvm through back constraints. In *Proc. of the International Conference on Machine Learning (ICML)*, 2006.

[14] L. Ljung. *System Identification*. Prentice hall, 1987.

[15] D. Nguyen-Tuong, M. Seeger, and J. Peters. Local Gaussian process regression for real time online model learning and control. In *Advances in Neural Information Processing Systems 22 (NIPS)*, 2008.

[16] C. Plagemann, D. Fox, and W. Burgard. Efficient failure detection on mobile robots using Gaussian process proposals. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[17] C.E. Rasmussen and C.K.I. Williams. *Gaussian processes for machine learning*. The MIT Press, 2005.

[18] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, September 2005. ISBN 0-262-20162-3.

[19] R. Urtasun, D. Fleet, and P. Fua. Gaussian process dynamical models for 3D people tracking. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[20] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Kluwer Academic Publishers, 1996.

[21] J. Wang, D. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2008.