Auton Robot (2009) 27: 75–90
DOI 10.1007/s10514-009-9119-x

# GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models

Jonathan Ko · Dieter Fox

Received: 10 November 2008 / Accepted: 3 May 2009 / Published online: 15 May 2009

**Abstract** Bayesian filtering is a general framework for recursively estimating the state of a dynamical system. Key components of each Bayes filter are probabilistic prediction and observation models. This paper shows how non-parametric Gaussian process (GP) regression can be used for learning such models from training data. We also show how Gaussian process models can be integrated into different versions of Bayes filters, namely particle filters and extended and unscented Kalman filters. The resulting GP-BayesFilters can have several advantages over standard (parametric) filters. Most importantly, GP-BayesFilters do not require an accurate, parametric model of the system. Given enough training data, they enable improved tracking accuracy compared to parametric models, and they degrade gracefully with increased model uncertainty. These advantages stem from the fact that GPs consider both the noise in the system and the uncertainty in the model. If an approximate parametric model is available, it can be incorporated into the GP, resulting in further performance improvements. In experiments, we show different properties of GP-BayesFilters using data collected with an autonomous micro-blimp as well as synthetic data.

**Keywords** Gaussian process · Bayesian filtering · Dynamic modelling · Machine learning · Regression

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

D. Fox
e-mail: fox@cs.washington.edu

## 1 Introduction

Estimating the state of a dynamic system is a fundamental problem in robotics. The most successful techniques for state estimation are Bayesian filters such as particle filters or extended and unscented Kalman filters (Thrun et al. 2005). Bayes filters recursively estimate posterior probability distributions over the state of a system. The key components of a Bayes filter are the prediction and observation models, which probabilistically describe the temporal evolution of the process and the measurements returned by the sensors, respectively.[1] Typically, these models are parametric descriptions of the involved processes. The parameters and noise components of the models can be estimated from training data or tuned manually (Abbeel et al. 2005; Bar-Shalom et al. 2001; Limketkai et al. 2007). Even though such parametric models are very efficient, accurate parametric models are difficult to obtain, and their predictive capabilities may be limited because they often ignore hard to model aspects of the process. Examples of difficult to model systems include anatomically correct robot hands (Deshpande et al. 2009), the dynamics of robotics arms (Nguyen and Peters 2008), or the spatial distribution of wireless signal strength (Ferris et al. 2006). A main problem is often that although the basic physics are easy to understand, these systems have very complex relationships between their various elements. In addition, some aspects of the system may be hard to directly measure, such as friction.

To overcome the limitations of parametric models, researchers have recently introduced non-parametric, Gaussian process regression models (Rasmussen and Williams

[1]Throughout this paper we will use the term "prediction" model instead of "process" model in order to avoid confusion with the term Gaussian process.

2005) to learn prediction and observation models for dynamical systems. GPs have been applied successfully to the problem of learning predictive state models (Girard et al. 2005; Grimes et al. 2006; Ko et al. 2007b; Liu et al. 2005). The fact that GP regression models provide uncertainty estimates for their predictions allows them to be readily incorporated into particle filters as observation models (Ferris et al. 2006) or for improved sampling distributions (Plagemann et al. 2007).

In this paper we demonstrate the integration of Gaussian Processes (GP) into different forms of Bayes filters. Specifically, we show how GP prediction and observation models can be combined with particle filters (GP-PF), extended Kalman filters (GP-EKF), and unscented Kalman filters (GP-UKF) (parts of this work have been published in Ko and Fox 2008; Ko et al. 2007a). The development of GP-EKFs requires a linearization of GP regression models, which we derive in this paper. The resulting GP-BayesFilters inherit the following features from GP regression:

- GP-BayesFilters do not depend on the availability of parametric prediction and observation models. The underlying models and all their parameters can be learned from training data, using non-parametric regression.
- GP-BayesFilters can take advantage of parametric models, if available. By incorporating such models into the GP regression, the filter parameters can typically be learned from significantly less training data.
- GP-BayesFilters generate state-dependent uncertainty estimates that take both noise and regression uncertainty due to limited training data into account. As a result, the filter automatically increases its uncertainty when the process enters areas in which not enough training data is available.
- GP-BayesFilters using standard GPs might become too inefficient when large training data sets are required. However, the incorporation of *sparse* GPs can significantly increase the efficiency of GP-BayesFilters. Furthermore, using heteroscedastic GPs results in GP-BayesFilters that can model systems with state dependent noise.

In addition to describing the formal framework of GP-BayesFilters, we perform a thorough comparison of the performance of the different filters based on simulation experiments and data collected by a robotic blimp.

This paper is organized as follows. After discussing related work, we provide the necessary background on Gaussian processes and Bayesian filtering. We then show how we can obtain GP prediction and observation models from data in Sect. 4. The different instantiations of GP-BayesFilters are introduced in Sect. 5, followed by an experimental evaluation. We conclude in Sect. 7.

## 2 Related work

The application of GPs to prediction modeling or dynamical system identification has been done by several researchers (Girard et al. 2005; Kocijan et al. 2003). Girard and colleagues use multiple previous states to make future predictions. The key novelty in this work is to not only consider the mean of the previous states, but also their uncertainty. They show how GPs learned using this type of data lead to more accurate predictions. The drawback is that it requires large amounts of training data which may not be available for highly complex, high dimensional systems. System identification using Gaussian processes is also addressed in (Grimes et al. 2006). In this case, Gaussian processes model the state dynamics of a humanoid robot. The focus of this work was on dynamic imitation of a human with a robot while considering external forces like gravity and inertia.

The use of GPs for observation models has also been considered. In (Ferris et al. 2007, 2006), GPs represent observation models for a person's or robot's position based on wireless signal strength. They develop a system for indoor localization using a particle filter. The fact that GP regression models provide uncertainty estimates allows them to be readily incorporated into the PF. GP observation models are again applied for localization with a particle filter in (Brooks et al. 2006). In this case, the observation model is constructed for an omni-directional camera. Dimensionality reduction is done on the camera images via wavelet decomposition. This data along with ground truth robot positions are used as the training data for the Gaussian process observation model.

Gaussian processes have been applied to various robotics applications. In (Plagemann et al. 2007), they are used to sense failure modes of a mobile robot. Online GPs are used to represent a forward dynamics model for several robotic arms in (Nguyen and Peters 2008). In (Engel et al. 2006), Gaussian processes are applied to reinforcement learning for mobile robots. GPs are employed to learn the value function through temporal difference learning. This technique is demonstrated on a simulated octopus arm. GPs are also applied to the reinforcement learning problem in (Ko et al. 2007b), where they are used to find parameters of a controller for a small blimp. RL episodes were run in simulation using a Gaussian process prediction model.

The idea of combining parametric and non-parametric models for regression has also been explored in (Sanner and Slotine 1991) where a neural network is combined with a standard parametric model in learning function approximations. However, their main focus is on online learning of the target function and theoretical guarantees of convergence.

A relevant extension of GPs was developed by Lawrence (2005), who used GPs for dimensionality reduction in a technique called Gaussian process latent variable model (GPLVM). This work is similar to PCA dimensionality reduction, except with a Gaussian process mapping between latent and data space. In related work, Wang et al. (2006) extend GPLVMs by learning a dynamics model in latent space. They demonstrate their technique on several different human motion patterns. An interesting connection to our work is that it is possible to extract a GP-BayesFilter from a GPDM, which is applicable even when ground truth system data is unavailable.

There is also a large amount of literature dedicated to improving Gaussian process efficiency and expressive power. These methods are somewhat orthogonal to our work since GP-BayesFilters can employ any type of GPs as long as they give adequate predictions and uncertainties. We test several of these algorithms in the experimental results of this paper. One of the main drawbacks of using Gaussian processes is the cubic complexity of training given the number of training points. In order to reduce this complexity, several researchers introduced *sparse* GP methods (Csató and Opper 2002; Seeger and Williams 2003; Seo et al. 2000; Smola and Bartlett 2001; Snelson and Ghahramani 2006). The idea of sparse GPs is to reduce the training set size by intelligently selecting a subset of training points, also called the active set, on which calculations are performed. Even though calculations are only done on the active training set, the full training data is taken into account by the formulation of these sparse GPs. A comprehensive summary of most of these techniques is presented in (Quinonero-Candela and Rasmussen 2005). In addition to sparse GPs, there are also modifications to GPs that allow them to model a wider array of phenomena. Work on modeling non-stationary processes, ones in which the smoothness of the process is state dependent, can be found in (Paciorek 2003) and (Plagemann et al. 2008). Finally, heteroscedastic processes, where noise is state dependent, can be modeled using techniques found in (Kersting et al. 2007) and (Goldberg et al. 1998).

## 3 Background for GP-BayesFilters

Before we describe the generic GP-BayesFilter, let us discuss the basic concepts underlying Gaussian processes and Bayes filters.

### 3.1 Gaussian processes for regression

Gaussian processes are a powerful, non-parametric tool for learning regression functions from sample data. Key advantages of GPs are their modeling flexibility, their ability to provide uncertainty estimates, and their ability to learn noise and smoothness parameters from training data (Rasmussen and Williams 2005).

#### *3.1.1 Basic Gaussian process model*

A Gaussian process represents distributions over functions based on training data. To see this, assume we have a set of training data, $D = \langle X, \mathbf{y}\rangle$, where $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$ is a matrix containing $d$-dimensional input examples $\mathbf{x}_i$, and $\mathbf{y} = [y_1, y_2, \dots, y_n]$ is a vector containing scalar training outputs $y_i$. One assumes that the observations are drawn from the noisy process

$$y_i = f(\mathbf{x}_i) + \epsilon, \tag{1}$$

where $\varepsilon$ is zero mean, additive Gaussian noise with variance $\sigma_n^2$. Conditioned on training data $D = \langle X, \mathbf{y}\rangle$ and a test input $\mathbf{x}_*$, a GP defines a Gaussian predictive distribution over the output $y_*$ with mean

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

and variance

$$\mathrm{GP}_\Sigma(\mathbf{x}_*, D) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T K^{-1}\mathbf{k}_*. \tag{3}$$

Here, $k$ is the kernel function of the GP, $\mathbf{k}_*$ is a vector defined by kernel values between $\mathbf{x}_*$ and the training inputs $X$, and $K$ is the $n \times n$ kernel matrix of the training input values; that is, $\mathbf{k}_*[i] = k(\mathbf{x}_*, \mathbf{x}_i)$ and $K[i,j] = k(\mathbf{x}_i, \mathbf{x}_j)$. Note that the prediction uncertainty, captured by the variance $\mathrm{GP}_\Sigma$, depends on both the process noise and the correlation between the test input and the training data.

The choice of the kernel function depends on the application, the most widely used being the squared exponential, or Gaussian, kernel with additive noise:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}')W(\mathbf{x}-\mathbf{x}')^T} + \sigma_n^2\delta, \tag{4}$$

where $\sigma_f^2$ is the signal variance. The signal variance basically controls the uncertainty of predictions in areas of low training data density. The diagonal matrix $W$ contains the length scales of the process, such that $W = \mathrm{diag}([1/l_1^2, 1/l_2^2, \dots, 1/l_d^2])$. The length scales reflect the relative smoothness of the process along the different input dimensions. The final GP parameter is $\sigma_n^2$, which is controls the global noise of the process. An example of GP regression is given in Fig. 1.

#### *3.1.2 Hyperparameter learning*

The parameters $\boldsymbol{\theta} = [W, \sigma_f, \sigma_n]$ are called the hyperparameters of the Gaussian process. They can be learned by maximizing the log marginal likelihood of the training outputs given the inputs:

$$\boldsymbol{\theta}_{max} = \underset{\theta}{\mathrm{argmax}} \{\log(p(\mathbf{y}|X, \boldsymbol{\theta}))\}. \tag{5}$$

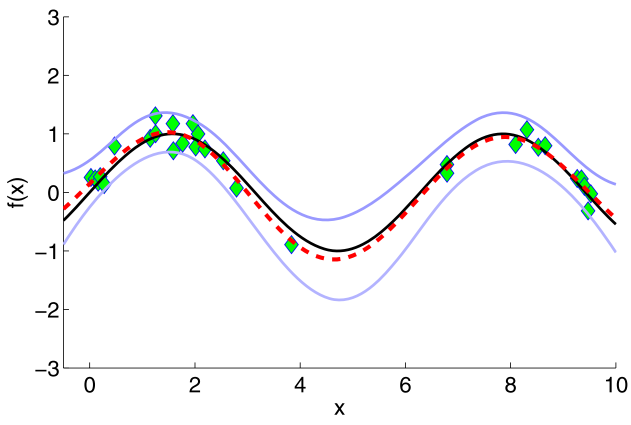

**Fig. 1** Example of one-dimensional GP regression. Shown are a sine function (*black*), noisy samples drawn from the function (*diamonds*), the resulting GP mean function estimate (*dashed*), and the GP uncertainty sigma bounds (*blue/gray*). The GP hyperparameters were determined via optimization of the data likelihood. Note how the uncertainty widens in the $x$-interval [3, 7] due to the sparseness of data points in this area

The log term in (5) can be expressed as

$$\log(p(\mathbf{y}|X,\boldsymbol{\theta})) = -\frac{1}{2}\mathbf{y}^T(K(X,X)+\sigma_n^2 I)^{-1}\mathbf{y} \\ -\frac{1}{2}\log|K(X,X)+\sigma_n^2 I| - \frac{n}{2}\log 2\pi. \tag{6}$$

This optimization problem can be solved using numerical optimization techniques such as conjugate gradient ascent (Rasmussen and Williams 2005). The partial derivatives of the log likelihood are required for this optimization and can be expressed as

$$\frac{\partial}{\partial\theta_k}\log(p(\mathbf{y}|X,\boldsymbol{\theta})) = \frac{1}{2}\operatorname{tr}\left[(K^{-1}\mathbf{y})(K^{-1}\mathbf{y})^T\frac{\partial K}{\partial\theta_k}\right]. \tag{7}$$

Each element $\frac{\partial K[i,j]}{\partial\theta_k}$ is the partial derivative of the kernel function with respect to the hyperparameters:

$$\frac{\partial k(\mathbf{x}_i,\mathbf{x}_j)}{\partial\sigma_f} = 2\sigma_f e^{-\frac{1}{2}(\mathbf{x}_i-\mathbf{x}_j)W(\mathbf{x}_i-\mathbf{x}_j)^T}, \tag{8}$$

$$\frac{\partial k(\mathbf{x}_i,\mathbf{x}_j)}{\partial\sigma_n} = 2\sigma_n\delta, \tag{9}$$

$$\frac{\partial k(\mathbf{x}_i,\mathbf{x}_j)}{\partial W_{ii}} = -\frac{1}{2}(\mathbf{x}_i[i]-\mathbf{x}_j[i])^2\sigma_f^2 e^{-\frac{1}{2}(\mathbf{x}_i-\mathbf{x}_j)W(\mathbf{x}_i-\mathbf{x}_j)^T}. \tag{10}$$

The optimization problem is non-convex, so there is no guarantee of finding a global optimum. However, the optimization nevertheless tends to work well in practice.

### 3.1.3 Sparse Gaussian processes

The efficiency of GPs can be greatly increased by reducing the number of data points used to represent the GP. While different algorithms have been introduced for generating such sparse GPs, we here focus on an approach that uses a set of so-called pseudo-inputs to represent the training data (Snelson and Ghahramani 2006). The key idea of this approach is that although the pseudo-inputs (active points) are initialized as a subset of the training points, they are continuous variables the values of which are determined via optimization. This enables the simultaneous optimization of both the GP hyperparameters and the locations of the pseudo-inputs. This optimization is smoother than that done in other sparse GP work such as (Csató and Opper 2002; Seeger and Williams 2003; Smola and Bartlett 2001), where active points are a subset of the training points. The sparse GP learning in these other works involves iteratively selecting points and optimizing the hyperparameters. The addition of new points interferes with hyperparameter optimization which might make convergence difficult. In general, the use of sparse GP techniques reduces training complexity from $O(n^3)$ to $O(m^2n)$ where $n$ and $m$ are the number of training and active points respectively. The complexity of mean prediction becomes $O(m)$ and covariance prediction $O(m^2)$.

### 3.1.4 State-dependent noise models

In (Kersting et al. 2007), Kersting and colleagues describe a technique for using GPs to model systems with state dependent noise. These systems are also known as heteroscedastic systems. This is opposed to the regular GP with a stationary kernel function which assumes a global noise value, and where all variations in the uncertainty are a result of differences in training data density. State dependent noise models have been examined before in other contexts. Techniques for modeling heteroscedastic systems have been considered using SVMs, splines, and locally linear regression models in (Edakunni et al. 2007), (Nott 2006), and (Schölkopf et al. 2000), respectively. This work is closely related to (Goldberg et al. 1998). Both use two separate GPs to model the data. The first GP learns to predict the mean in much the same way the regular GP does. The other GP is used to model the prediction uncertainty. The key difference is that Kersting et al. use the most likely noise instead of a MCMC approximation of the noise distribution. Using this simplification, they can develop an EM-like algorithm for optimizing the model. Since these techniques remains fully within GP regression framework, integration with sparse GP techniques is possible. We examine GP-BayesFilters with these GP models in Sect. 6.

## 3.2 Bayes Filters

Bayes filters recursively estimate posterior distributions over the state $\mathbf{x}_k$ of a dynamical system conditioned on all sensor information collected so far:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k},\mathbf{u}_{1:k-1}) \propto p(\mathbf{z}_k|\mathbf{x}_k)\int p(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k})d\mathbf{x}_{k-1}. \quad (11)$$

Here $\mathbf{z}_{1:k}$ and $\mathbf{u}_{1:k-1}$ are the histories of sensor measurements and controls obtained up to time $k$. The term $p(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_{k-1})$ is the *prediction model*, a probabilistic model of the system dynamics. $p(\mathbf{z}_k|\mathbf{x}_k)$, the *observation model*, describes the likelihood of making an observation $\mathbf{z}_k$ given the state $\mathbf{x}_k$. Typically, these models are parametric descriptions of the underlying processes, see (Thrun et al. 2005) for several examples. In GP-BayesFilters, both prediction and observation models are learned from training data using non-parametric, Gaussian process regression.

In this paper we implement and test three different instantiations of Bayes filters using GP prediction and observation models. Specifically, we present algorithms for GP integration into particle filters (PF), extended Kalman filters (EKF), and unscented Kalman filters (UKF). These filters apply different approximations to posteriors over the state space.

The key idea of the particle filter is to represent posteriors by sets of weighted samples, or particles. These filters are highly accurate given enough particles. This accuracy comes at a price, however, since the particle filter may require a large number of particles and thus suffers from poor time efficiency. This is mitigated somewhat by using Rao-Blackwellised PFs (Doucet et al. 2000).

Kalman filters, on the other hand, represent the state estimate with a Gaussian distribution. This allows for highly efficient updates but at a cost of representational power and hence accuracy. For example, Kalman filters are incapable of representing multimodal distributions. Extended and unscented Kalman filters are extensions of regular Kalman filters designed to work with non-linear systems. In order to do this, both extended and unscented Kalman filters linearize the non-linear prediction and observation functions. At the core of the Kalman filter is the prediction of the new state from the previous state, and the prediction of the observation from that new state. The extended Kalman filter uses the prediction and observation function to propagate the mean directly, and the Jacobian of the prediction and observation function to calculate the new covariance. The unscented Kalman filter propagates via the unscented transform. In it, a deterministic set of samples called *sigma* points are propagated using the prediction and observation function. The unscented Kalman filter then calculates the new mean and covariance from these sigma points. The UKF is in theory at least as accurate as the EKF (Julier and Uhlmann 1997), but we show later that this is not always the case when UKFs are integrated with GP prediction and observation models.

## 4 Learning prediction and observation models with GPs

Gaussian process regression can be applied directly to the problem of learning prediction and observation models required by the Bayes filter (11). In our context, a model needs to provide both a mean prediction as well as an uncertainty or noise in that prediction. Gaussian processes neatly provide both. There is an assumption made that the noise is Gaussian, but that is assumed by the Kalman filters as well. Particle filters can utilize non-Gaussian noise models, so this may be a slight drawback of using GP models.

### 4.1 Training data

The training data is a sampling from the dynamics and observations of the system. We make the assumption that it is representative of the system, that is, the training data covers those parts of the state space that are visited during normal operation (we explore the behavior of GP models when this is not the case in Sect. 6.3.1.) The training data for each GP consists of a set of input-output relations. The prediction model maps the state and control, $(\mathbf{x}_k,\mathbf{u}_k)$, to the state transition $\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$. The next state can then be found by adding the state transition to the previous state. The observation model maps from the state, $\mathbf{x}_k$, to the observation, $\mathbf{z}_k$. The appropriate form of the prediction and observation training data sets is thus

$$D_p = \langle (X,U), X' \rangle, \quad (12)$$

$$D_o = \langle X, Z \rangle, \quad (13)$$

where $X$ is a matrix containing ground truth states, and $X' = [\Delta\mathbf{x}_1, \Delta\mathbf{x}_2, \ldots, \Delta\mathbf{x}_k]$ is a matrix containing transitions made from those states when applying the controls stored in $U$. $Z$ is the matrix of observations made when in the corresponding states $X$.

The resulting GP prediction and observation models are then

$$p(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_{k-1}) \approx \mathcal{N}\left(\mathrm{GP}_\mu([\mathbf{x}_{k-1},\mathbf{u}_{k-1}],D_p), \mathrm{GP}_\Sigma([\mathbf{x}_{k-1},\mathbf{u}_{k-1}],D_p)\right) \quad (14)$$

and

$$p(\mathbf{z}_k|\mathbf{x}_k) \approx \mathcal{N}\left(\mathrm{GP}_\mu(\mathbf{x}_k,D_o), \mathrm{GP}_\Sigma(\mathbf{x}_k,D_o)\right), \quad (15)$$

respectively. The reader may notice that while these models are Gaussians, both the means and variances are non-linear functions of the input and training data. Furthermore, the locally Gaussian nature of these models allows a very natural integration into different instantiations of Bayes filters, as we will describe in Sect. 5.

GPs are typically defined for scalar outputs, and GP-BayesFilters represent models for vectorial outputs by learning a different GP for each output dimension. Since the output dimensions are now independent of each other, the resulting noise covariances $\mathrm{GP}_\Sigma$ are diagonal matrices.

### 4.2 Model enhancement using parametric models

So far, the evolution of the dynamic system and the observation predictions are represented by Gaussian processes alone. There are some drawbacks to this approach. Because there is a zero mean assumption in the GP, test states that are far away from the training states will have outputs that quickly tend towards zero. This makes the choice of training data for the GP very important. In addition, purely non-parametric models lack interpretability. Since they are data-driven and lack parameters, non-parametric models give no insight into the system.

Higher prediction accuracy can be obtained by combining GP models with parametric models. A parametric model is one which attempts to represent a particular phenomenon with physical equations. For example a blimp model would be described by aerodynamics equations, a mobile robot with physics or inverse kinematics. These equations leave open variables or parameters which are optimized with respect to training data. The noise component of these models can be found by calculating the expected squared difference between the parametric function and the ground truth. The disadvantage of parametric models is that substantial domain expertise is required to build these models, and even then they are often simplified representations of the actual systems.

A combination of GP and parametric models alleviates some of the problems with either model alone. First, there is an overall increase in accuracy, since the GP model can represent aspects of the system that are not captured by the parametric model. Also, since parametric models capture the physical process underlying the dynamical system, they typically generalize well beyond the training data. We call the combination of GP and parametric models enhanced-GP (EGP) models.

Essentially, EGPs learn the residual output after factoring the contributions of the parametric model. For the prediction model, instead of learning the change in state, we need to learn the change in state after processing by the parametric model. If $f(\mathbf{x}, \mathbf{u})$ is the parametric mean function which predicts the change in system state from time $k$ to $k+1$, then the training for the enhanced-GP model becomes

$$\Delta\hat{\mathbf{x}}_k = \mathbf{x}_{k+1} - \mathbf{x}_k - f(\mathbf{x}_k, \mathbf{u}_k). \tag{16}$$

The training input for the GP remains the same. The training data for the prediction model is now

$$D_p = \langle (X, U), \hat{X}' \rangle, \tag{17}$$

where $\hat{X}' = [\Delta\hat{\mathbf{x}}_1, \Delta\hat{\mathbf{x}}_2, \ldots, \Delta\hat{\mathbf{x}}_k]$. The training output for the observation model can be derived similarly. It is

$$\hat{\mathbf{z}}_k = \mathbf{z}_{k+1} - \mathbf{z}_k - g(\mathbf{x}_k), \tag{18}$$

where $g(\mathbf{x})$ is the parametric observation function and $\hat{Z} = [\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_k]$. The training data for the enhanced-GP observation model becomes

$$D_o = \langle X, \hat{Z} \rangle. \tag{19}$$

## 5 Instantiations of GP-BayesFilters

We will now show how GP models can be incorporated into different instantiations of Bayes filters. For notation, we will stick close to the versions presented in (Thrun et al. 2005).

### 5.1 GP-PF: Gaussian process particle filters

Particle filters are sample-based implementations of Bayes filters which represent posteriors over the state $\mathbf{x}_k$ by sets $\mathcal{X}_k$ of weighted samples:

$$\mathcal{X}_k = \{\langle \mathbf{x}_k^m, w_k^{(m)} \rangle \mid m = 1, \ldots, M\}.$$

Here each $\mathbf{x}_k^m$ is a sample (or state), and each $w_k^{(m)}$ is a non-negative numerical factor called *importance weight*. Particle filters update posteriors according to a sampling procedure (Thrun et al. 2005). Table 1 shows how this procedure can be implemented with GP prediction and observation models. In Step 4, the state at time $k$ is sampled based on the previous state $\mathbf{x}_{k-1}^m$ and control $\mathbf{u}_{k-1}$, using the GP prediction model defined in (14). Here, $\mathrm{GP}([\mathbf{x}_{k-1}^m, \mathbf{u}_{k-1}], D_p)$ is short for the Gaussian represented by $\mathcal{N}(\mathrm{GP}_\mu([\mathbf{x}_{k-1}^m, \mathbf{u}_{k-1}], D_p),\ \mathrm{GP}_\Sigma([\mathbf{x}_{k-1}^m, \mathbf{u}_{k-1}], D_p))$. Note that the covariance of this prediction is typically different for each sample, taking the local density of training data into account. Importance sampling is implemented in Step 5, where each particle is weighted by the likelihood of the actual most recent measurement $\mathbf{z}_k$ given expected measurement from the sampled state $\mathbf{x}_k^m$. This likelihood can be easily extracted from the GP observation model defined in (15). All other steps are identical to the generic particle filter algorithm, where Steps 8 through 11 implement the resampling step (see Thrun et al. 2005).

**Table 1** The GP-PF algorithm

```
1:  Algorithm GP-PF($\mathcal{X}_{k-1}, \mathbf{u}_{k-1}, \mathbf{z}_k$):
2:  $\hat{\mathcal{X}}_k = \mathcal{X}_k = \emptyset$
3:  for $m = 1$ to $M$ do
4:      sample $\mathbf{x}_k^m \sim \mathbf{x}_{k-1}^m + \mathrm{GP}([\mathbf{x}_{k-1}^m, \mathbf{u}_{k-1}], D_p)$
5:      $w_k^{[m]} = \mathcal{N}(\mathbf{z}_k; \mathrm{GP}_\mu(\mathbf{x}_k^m, D_o), \mathrm{GP}_\Sigma(\mathbf{x}_k^m, D_o))$
6:      add $\langle \mathbf{x}_k^m, w_k^{[m]} \rangle$ to $\hat{\mathcal{X}}_k$
7:  endfor
8:  for $m = 1$ to $M$ do
9:      draw $i$ with probability $\propto w_k^{[i]}$
10:     add $\mathbf{x}_k^{[i]}$ to $\mathcal{X}_k$
11: endfor
12: return $\mathcal{X}_k$
```

**Table 2** The GP-EKF algorithm

```
1:  Algorithm GP-EKF($\mu_{k-1}, \Sigma_{k-1}, \mathbf{u}_{k-1}, \mathbf{z}_k$):
2:  $\hat{\mu}_k = \mu_{k-1} + \mathrm{GP}_\mu([\mu_{k-1}, \mathbf{u}_{k-1}], D_p)$
3:  $Q_k = \mathrm{GP}_\Sigma([\mu_{k-1}, \mathbf{u}_{k-1}], D_p)$
4:  $G_k = I + \frac{\partial \mathrm{GP}_\mu([\mu_{k-1}, \mathbf{u}_{k-1}], D_p)}{\partial \mathbf{x}_{k-1}}$
5:  $\hat{\Sigma}_k = G_k \, \Sigma_{k-1} \, G_k^T + Q_k$
6:  $\hat{\mathbf{z}}_k = \mathrm{GP}_\mu(\hat{\mu}_k, D_o)$
7:  $R_k = \mathrm{GP}_\Sigma\left(\hat{\mu}_k, D_o\right)$
8:  $H_k = \frac{\partial \mathrm{GP}_\mu(\hat{\mu}_k, D_o)}{\partial \mathbf{x}_k}$
9:  $K_k = \hat{\Sigma}_k \, H_k^T (H_k \, \hat{\Sigma}_k \, H_k^T + R_k)^{-1}$
10: $\mu_k = \hat{\mu}_k + K_k(\mathbf{z}_k - \hat{\mathbf{z}}_k)$
11: $\Sigma_k = (I - K_k \, H_k) \, \hat{\Sigma}_k$
12: return $\mu_k, \Sigma_k$
```

### 5.2 GP-EKF: Gaussian process extended Kalman filters

The following describes the integration of GP prediction and observation models into the extended Kalman filter. In addition to the GP mean and covariance estimates used in the GP-PF, the incorporation of GP models into the EKF requires a linearization of the GP prediction and observation model in order to propagate the state and observation, respectively. For the EKF, this linearization is computed by taking the first term of the Taylor series expansion of the GP function. The Jacobian of the GP mean function (2) can be expressed as

$$\frac{\partial(\mathrm{GP}_\mu(\mathbf{x}_*, D))}{\partial(\mathbf{x}_*)} = \frac{\partial(\mathbf{k}_*)}{\partial(\mathbf{x}_*)}^T K^{-1}\mathbf{y}. \tag{20}$$

As noted above, $\mathbf{k}_*$ is the vector of kernel values between the query input $\mathbf{x}_*$ and the training inputs $X$. The Jacobian of the kernel vector function with respect to the inputs is

$$\frac{\partial(\mathbf{k}_*)}{\partial(\mathbf{x}_*)} = \begin{bmatrix} \frac{\partial(k(\mathbf{x}_*,\mathbf{x_1}))}{\partial(\mathbf{x}_*[1])} & \cdots & \frac{\partial(k(\mathbf{x}_*,\mathbf{x_1}))}{\partial(\mathbf{x}_*[d])} \\ \vdots & \ddots & \vdots \\ \frac{\partial(k(\mathbf{x}_*,\mathbf{x_n}))}{\partial(\mathbf{x}_*[1])} & \cdots & \frac{\partial(k(\mathbf{x}_*,\mathbf{x_n}))}{\partial(\mathbf{x}_*[d])} \end{bmatrix}, \tag{21}$$

where $n$ is the number of training points and $d$ is the dimensionality of the input space. The actual partial derivatives depend on the kernel function used. For the squared exponential kernel, it is

$$\frac{\partial(k(\mathbf{x}_*, \mathbf{x}))}{\partial(\mathbf{x}_*[i])} = -W_{ii}(\mathbf{x}_*[i] - \mathbf{x}[i])\sigma_f^2 e^{-\frac{1}{2}(\mathbf{x}_*-\mathbf{x})W(\mathbf{x}_*-\mathbf{x})^T}. \tag{22}$$

Equation (20) defines the $d$-dimensional Jacobian vector for the GP mean function for a single output dimension. The full $d \times d$ Jacobian of a prediction or observation model is determined by stacking $d$ Jacobian vectors together, one for each of the output dimensions.

We are now prepared to incorporate GP prediction and observation models into an EKF, as shown in Table 2. Step 2 uses the GP prediction model (2) to generate the predicted mean $\hat{\mu}_k$. Step 3 sets the additive process noise, $Q_k$, which corresponds directly to the GP uncertainty. $G_k$, the linearization of the prediction model, is the sum of the identity matrix and the Jacobian of the GP mean function found in (20). The identity matrix is necessary since the GP mean function only represents the change in state from one time step to the other. A change in any of the dimensions of the previous state has a direct effect on the new state. Step 5 uses these matrices to compute the predictive uncertainty, in the same way as the standard EKF algorithm. Similarly, Steps 6 through 8 compute the predicted observation, $\hat{\mathbf{z}}_k$, the noise covariance, $R_k$, and the linearization of the observation model, $H_k$, using the GP observation model. The remaining steps are identical to the standard EKF algorithm, where Step 9 computes the Kalman gain, followed by the update of the mean and covariance estimates in Steps 10 and 11, respectively.

### 5.3 GP-UKF: Gaussian process unscented Kalman filters

The GP-UKF algorithm is shown in Table 3. The key idea underlying unscented Kalman filters is to replace the linearization employed by the EKF by a more accurate linearization based on the unscented transform. To do so, the UKF generates a set of so-called sigma points based on the mean and variance estimates of the previous time step. This set, generated in Step 2, contains $2d + 1$ points, where $d$ is the dimensionality of the state space. Each of these points is then projected forward in time using the GP prediction model in Step 3. The additive process noise, computed in Step 4, is identical to the noise used in the GP-EKF algorithm.[2] Steps 5 and 6 are identical to the standard unscented

[2]While a more accurate estimate of the uncertainty $Q_k$ could be computed by taking into account the GP uncertainty at each sigma point,

**Table 3** The GP-UKF algorithm

```
1:  Algorithm GP-UKF($\mu_{k-1}, \Sigma_{k-1}, \mathbf{u}_{k-1}, \mathbf{z}_k$):
2:  $\mathcal{X}_{k-1} = (\mu_{k-1} \quad \mu_{k-1} + \gamma\sqrt{\Sigma_{k-1}} \quad \mu_{k-1} - \gamma\sqrt{\Sigma_{k-1}})$
3:  for $i = 0 \ldots 2n$: $\bar{\boldsymbol{\mathcal{X}}}_k^{[\mathbf{i}]} = \mathcal{X}_{k-1}^{[i]} + \mathrm{GP}_\mu([\mathcal{X}_{k-1}^{[i]}, \mathbf{u}_{k-1}], D_p)$
4:  $Q_k = \mathrm{GP}_\Sigma\left([\mu_{k-1}, \mathbf{u}_{k-1}], D_p\right)$
5:  $\hat{\mu}_k = \sum_{i=0}^{2n} w_m^{[i]} \bar{\boldsymbol{\mathcal{X}}}_k^{[\mathbf{i}]}$
6:  $\hat{\Sigma}_k = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\boldsymbol{\mathcal{X}}}_k^{[\mathbf{i}]} - \hat{\mu}_k)(\bar{\boldsymbol{\mathcal{X}}}_k^{[\mathbf{i}]} - \hat{\mu}_k)^T + Q_k$
7:  $\hat{\boldsymbol{\mathcal{X}}}_k = (\hat{\mu}_t \quad \hat{\mu}_t + \gamma\sqrt{\hat{\Sigma}_k} \quad \hat{\mu}_t - \gamma\sqrt{\hat{\Sigma}_k})$
8:  for $i = 0 \ldots 2n$: $\hat{\boldsymbol{\mathcal{Z}}}_k^{[\mathbf{i}]} = \mathrm{GP}_\mu(\hat{\boldsymbol{\mathcal{X}}}_k^{[\mathbf{i}]}, D_o)$
9:  $R_k = \mathrm{GP}_\Sigma(\hat{\mu}_k, D_o)$
10: $\hat{z}_k = \sum_{i=0}^{2n} w_m^{[i]} \hat{\boldsymbol{\mathcal{Z}}}_k^{[\mathbf{i}]}$
11: $S_k = \sum_{i=0}^{2n} w_c^{[i]} (\hat{\boldsymbol{\mathcal{Z}}}_k^{[\mathbf{i}]} - \hat{z}_k)(\hat{\boldsymbol{\mathcal{Z}}}_k^{[\mathbf{i}]} - \hat{z}_k)^T + R_k$
12: $\hat{\Sigma}_k^{x,z} = \sum_{i=0}^{2n} w_c^{[i]} (\hat{\boldsymbol{\mathcal{X}}}_k^{[\mathbf{i}]} - \hat{\mu}_k)(\hat{\boldsymbol{\mathcal{Z}}}_k^{[\mathbf{i}]} - \hat{z}_k)^T$
13: $K_k = \hat{\Sigma}_k^{x,z} \, S_k^{-1}$
14: $\mu_k = \hat{\mu}_k + K_k(z_k - \hat{z}_k)$
15: $\Sigma_k = \hat{\Sigma}_k - K_k \, S_k \, K_k^T$
16: return $\mu_k, \Sigma_k$
```

Kalman filter; they compute the predictive mean and covariance from the predicted sigma points. A new set of sigma points is extracted from this updated estimate in Step 7. The GP observation model is used in Step 8 to predict an observation for each of these points. The observation noise matrix, $R_k$, is set to the GP uncertainty in Step 9.

Steps 10 through 16 are standard UKF updates (see Thrun et al. 2005). The mean observation is determined from the observation sigma points in Step 10, and Steps 11 and 12 compute uncertainties and correlations used to determine the Kalman gain in Step 13. Finally, the next two steps update the mean and covariance estimates, which are returned in Step 16.

### 5.4 GP-BayesFilter complexity analysis

This section examines the run-time complexity of the three different algorithms. In typical tracking applications, the number $n$ of training points used for the GP is much higher than the dimensionality $d$ of the state space. In these cases, the complexity of GP-BayesFilters is dominated by GP operations, on which we will concentrate our analysis. Furthermore, since the ratio between prediction and observation evaluations is the same for all algorithms, we focus on the prediction step of each algorithm (the correction step is analogous). $C$ denotes the cost of the various steps. The algorithms are broken down into the cost of kernel evaluations, $C_{\text{kern}}$, and multiplications, $C_{\text{mult}}$.

Since GP-PFs need to perform one GP mean and variance computation per particle, the overall complexity of GP-PFs follows as

$$C_{\text{pf}} = M\,(C_\mu + C_\Sigma), \tag{23}$$

where $M$ is the number of particles. The GP-EKF algorithm requires one GP mean and variance computation plus one GP Taylor series expansion step:

$$C_{\text{ekf}} = C_\mu + C_\Sigma + C_{\text{tse}}. \tag{24}$$

The GP-UKF algorithm requires one GP mean computation for each sigma point. One GP variance computation is also necessary per step. Since $2d + 1$ sigma points are used for the prediction step, the complexity follows as

$$C_{\text{ukf}} = (2d + 1)C_\mu + C_\Sigma. \tag{25}$$

To get a more accurate measure, we have to consider the complexity of the individual GP operations. The core of each GP mean prediction operation consists of a kernel evaluation (4) followed by a multiplication. The computation of the mean function $\mathrm{GP}_\mu(\mathbf{x}_*, D)$ defined in (2) requires $n$ such evaluations, one for each combination of the query point $\mathbf{x}_*$ and a training point $\mathbf{x}_i$. In addition, this operation must be done for each output dimension since we use a separate GP for each dimension. We assume the number of output dimensions is equal to the dimensionality of the state space:

$$C_\mu = nd(C_{\text{kern}} + C_{\text{mult}}). \tag{26}$$

Note that the term $K^{-1}\mathbf{y}$ in (2) and (20) is independent of $\mathbf{x}_*$ and can be cached.

In computing the prediction covariance $\mathrm{GP}_\Sigma(\mathbf{x}_*, D)$, we assume the mean prediction has already been performed since this would be the case for all three GP-BayesFilter algorithms examined here. In particular, $\mathbf{k}_*$ can be assumed to be already calculated. Evaluation of $\mathbf{k}_*^T K^{-1}\mathbf{k}_*$ requires $n(d+1)$ multiplications for each output dimension. $k(\mathbf{x}_*, \mathbf{x}_*)$ is just a single kernel evaluation, so the total complexity is

$$C_\Sigma = nd(d+1)\,C_{\text{mult}} + d\,C_{\text{kern}}, \tag{27}$$

Finally, the linearization of the GP, given in (22), requires calculation of the Jacobian of the kernel function followed by a multiplication by $K^{-1}\mathbf{y}$. For the squared exponential kernel, each element of the Jacobian can be found after two

we here consider the more simple approach of only using the uncertainty at the mean sigma point.

additional multiplications, since the partial derivative can be rewritten as

$$\frac{\partial(k(\mathbf{x}_*, \mathbf{x}))}{\partial(\mathbf{x}_*[i])} = -W_{ii}(\mathbf{x}_*[i] - \mathbf{x}[i])k(\mathbf{x}_*, \mathbf{x}). \tag{28}$$

The kernel evaluations, $k(\mathbf{x}_*, \mathbf{x}_i)$, are cached, so the cost to compute the Jacobian of the kernel function for each output dimension is $2ndC_{\text{mult}}$. This is followed by multiplication with $K^{-1}\mathbf{y}$ which costs $nd\ C_{\text{mult}}$ per dimension. The overall cost of the linearization operation is then,

$$C_{\text{tse}} = 3nd^2\ C_{\text{mult}}. \tag{29}$$

The total costs in terms of kernel computations and multiplications are shown in Table 4.

This analysis has been done on the GP-BayesFilter using the squared exponential kernel. Unfortunately, the cost of kernel evaluations and multiplications cannot be unified into a single value since the exponential in the kernel function has variable cost depending on the hardware used. However, there are still some conclusions that can be made from this analysis. The first is GP-PF has very high computational cost since the number of particles required is often much greater than the dimensionality of the state space. However, GP-PF has a redeeming characteristic as it is the only filter that can represent multi-modal beliefs. GP-EKF has some more multiply operations than GP-UKF but they are on the same order. However, GP-UKF is $O(nd^2)$ in terms of kernel evaluations, while GP-EKF is only $O(nd)$. Since the kernel computation is generally much slower than a multiply operation, the kernel computation will tend to dominate in the run time for these two filters. GP-EKF will thus be significantly faster than GP-UKF. Results in Sect. 6 show this to be true.

**Table 4** Computational requirements

| | $C_{\text{kern}}$ | $C_{\text{mult}}$ |
|---|---|---|
| *GP-UKF* | $nd(2d+1)+d$ | $3nd(d+\frac{2}{3})$ |
| *GP-EKF* | $d(n+1)$ | $4nd(d+\frac{1}{2})$ |
| *GP-PF* | $Mnd$ | $Mnd(d+1)$ |

## 6 Experiments

We present two sets of experiments. In the first set, we evaluate different properties of GP-BayesFilters in the context of tracking a small blimp using cameras mounted in an indoor environment. The second set provides a setting in which GP-EKF may offer higher accuracy than GP-UKF.

### 6.1 Robotic blimp testbed

The experimental testbed for evaluating the GP-BayesFilters is a robotic micro-blimp flying through a motion capture lab, as shown in Fig. 2.

#### *6.1.1 Hardware*

A custom-built gondola is suspended beneath a 5.5 foot (1.7 meter) long envelope. The gondola houses two main fans that pivot together to provide thrust in the longitudinal (forwards-up) plane. A third motor located in the tail provides thrust to yaw the blimp about the body-fixed $Z$-axis. There are a total of three control inputs: the power of the gondola fans, the angle of the gondola fans, and the power of the tail fan.

A necessary component of GP-BayesFilters is ground truth data of the system. The system states, observations, and control inputs are all required in order to learn the GP prediction and observation models. For this experiment, observations for the filters come from two network cameras mounted in the laboratory. These are Panasonic model KX-HCM270 running at 1 Hertz each. Observations are formed by background subtraction followed by the fitting of an ellipse to the remaining (foreground) pixels. The observations are the parameters of the ellipse in image space, parametrized by the position, size, and orientation.

A VICON motion capture system captures the ground truth states of the blimp as it flies. The system tracks reflective markers attached to the blimp as 3D points in space, and uses a blimp model to convert marker positions to position and orientation of the blimp at each timestep. The velocity and rotational velocity can be obtained via a smoothed

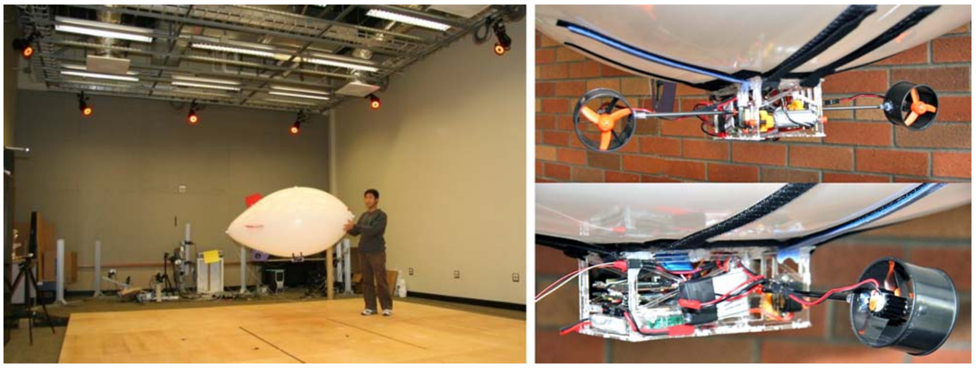
**Fig. 2** The *left image* shows the blimp used in our test environment equipped with a motion capture system. It has a customized gondola (*right images*) that includes an XScale based computer with sensors, two ducted fans that can be rotated by 360 degrees, and a webcam

calculation of the slope between sample points. The system runs at 120 Hz, but downsampled to 4 Hz to better match the cyclic rate of our cameras. In addition, the downsampled rate allows us to learn GPs that are capable of running in real time.

The final component of the ground truth data is the control inputs which are obtained from a human controlled joystick. These control inputs are then sent to the blimp. The data collected is then used to train the GPs and parametric models, and to evaluate the tracking performance of the various filtering algorithms. The data acquisition and integration code is written primarily in C++ whereas the numerical calculations are performed almost exclusively in MATLAB. Gaussian process code is from Neil Lawrence (http://www.dcs.shef.ac.uk/~neil/fgplvm/). All experiments were performed on an Intel® Xeon$^{\text{TM}}$ running at 3.40 GHz.

### 6.1.2 *Parametric prediction and observation models*

The parametric motion model appropriate for the blimp was described in detail in (Ko et al. 2007b). The state of the blimp consists of position $\mathbf{p}$, orientation $\boldsymbol{\xi}$ parametrized by Euler angles, linear velocity $\mathbf{v}$, and angular velocity $\boldsymbol{\omega}$. The resulting model has the form

$$\frac{d}{dt}\begin{bmatrix}\mathbf{p}\\ \boldsymbol{\xi}\\ \mathbf{v}\\ \boldsymbol{\omega}\end{bmatrix} = \begin{bmatrix} R(\boldsymbol{\xi})^e_b\mathbf{v} \\ H(\boldsymbol{\xi})\boldsymbol{\omega} \\ M^{-1}(\sum \text{Forces} - \boldsymbol{\omega}\times M\mathbf{v}) \\ J^{-1}(\sum \text{Torques} - \boldsymbol{\omega}\times J\boldsymbol{\omega})\end{bmatrix}. \qquad (30)$$

Here, $M$ is the mass matrix, $J$ is the inertia matrix, $R^e_b$ is the rotation matrix from body-fixed to inertial reference, and $H$ is the Euler kinematical matrix. The sum of forces and torques accounts for thrust produced by each motor, drag effects, and gravity. This parametric model is an ODE which must be solved in order to produce a function $f$ which predicts the next state given the current state and control input. $f$ is then a function that is discrete in time and of the same form as the GP prediction model mean function.

There are 16 parameters to the model which are learned by minimizing point-wise squared differences between simulated states and ground truth. Ground truth states for blimp motion comes from the VICON system. This parameter estimation is done in two stages. First, we perform maneuvers with the blimp in order to isolate and get approximate values for particular parameters. For example, to get an approximate value for the center of mass, the blimp can be rotated about the $X$ and $Y$ axes and released. Due to the blimp's self stabilizing feature, the point that moves the least is therefore a good approximation for the center of mass. Next, the parameters can then be optimized together in order to minimize the error between prediction and the training data using numerical techniques like conjugate gradient descent. This two step procedure alleviates problems with local minima.

**Table 5** Prediction model quality

| Propagation method | $p$ (mm) | $\xi$ (deg) | $v$ (mm/s) | $\omega$ (deg/s) |
|---|---|---|---|---|
| *Param* | $3.3 \pm 0.003$ | $0.5 \pm 0.0004$ | $14.6 \pm 0.01$ | $1.5 \pm 0.001$ |
| *GP* | $1.8 \pm 0.004$ | $0.2 \pm 0.0004$ | $9.8 \pm 0.02$ | $1.1 \pm 0.002$ |
| *EGP* | $1.6 \pm 0.004$ | $0.2 \pm 0.0005$ | $9.6 \pm 0.02$ | $1.3 \pm 0.003$ |

The model provides a thorough approximation of the true dynamics of the blimp, and has the same form as other models of rigid body dynamics (Gomes and Ramos 1998; Stevens and Lewis 1992). However, there are several approximations and simplifications that are made with this model. For example, the drag terms are linear, which may be appropriate for objects moving at low speeds, but unlikely to be fully accurate. Also, the is no attempt to model lift. The blimp can generate lift not only from the fixed stabilizer fins, but also from the body of the blimp itself. These effects can be added to the model, but would add to the complexity, and may require additional facilities to model, i.e. wind tunnel, etc. The lesson here is that learning a highly accurate parametric model is difficult even with expert domain knowledge. The GP model can and does show better performance in modeling blimp dynamics as can be seen in the next section.

The parametric observation model takes as input the six-dimensional pose of the blimp within the camera's coordinate system. Using computer vision techniques, the blimp axes are then projected into image space. Geometric calculations are used to find the parameters of an ellipse based on these projected axes. This observation model is only an approximation as it does not consider several factors, such as the barrel distortion of the camera. This model, however, does works well in practice.

## 6.2 Comparison of prediction and observation models

This first experiment is designed to test the quality of motion and observation models, which are the crucial components for Bayesian filtering. In total, three different ways of modeling the blimp motion and the camera observations are compared: parametric models (*Param*), GP only (*GP*), and Enhanced-GP (*EGP*). The training data for the blimp motion model was collected by flying the blimp manually via remote-control. To ensure even coverage of the state space, the training data for the observation model was collected by moving the blimp manually through the room. Both the GP prediction and observation model use roughly 900 training samples each.

The results for the prediction models are summarized in Table 5. The results are obtained as follows: start with a ground truth state, predict using the model, take the difference between the predicted state and the next ground truth

**Table 6** Observation model prediction quality

| Propagation method | position (px) | width (px) | height (px) | orientation (rad) |
|---|---|---|---|---|
| *Param* | 7.1 ± 0.0021 | 2.9 ± 0.0016 | 5.6 ± 0.0030 | 9.2 ± 0.0100 |
| *GP* | 4.7 ± 0.0022 | 3.2 ± 0.0019 | 1.9 ± 0.0012 | 9.1 ± 0.0088 |
| *EGP* | 3.9 ± 0.0021 | 2.4 ± 0.0018 | 1.9 ± 0.0014 | 9.4 ± 0.0099 |

**Table 7** Aggregate RMS percentage improvement over baseline parametric filter (higher values are better)

| | *GP* | *EGP* | *hetGP* | *sparseGP* |
|---|---|---|---|---|
| *UKF* | 30.75 ± 1.41 | 34.10 ± 1.76 | 35.76 ± 1.61 | 32.05 ± 2.02 |
| *EKF* | 27.66 ± 1.04 | 31.44 ± 2.43 | 33.70 ± 2.09 | 29.72 ± 1.90 |
| *PF* | 33.93 ± 7.24 | 35.95 ± 6.91 | na | 38.92 ± 2.17 |

state. Quality here is measured in terms of the average norm prediction error in position, $\mathbf{p}$, orientation, $\boldsymbol{\xi}$, forward velocity, $\mathbf{v}$, and angular velocity, $\boldsymbol{\omega}$. As can be seen, both GP-based models produce very similar results that are significantly better than the parametric approach.

The results for the observation models are summarized in Table 6. The results are obtained by comparing the difference between the ground truth observations with those found using the model. Quality is measured in mean norm error in ellipse position, width, height, and orientation. Here again, both *GP* and *EGP* outperform the parametric model. Note that the *EGP* error in orientation is surprisingly high. A further investigation showed that this result is not a due to a weakness of the *EGP* approach but rather an artifact of the Euler angle representation chosen in this experiment.

## 6.3 Tracking performance

Experimental results for tracking were obtained via 4-fold cross-validation. The data was broken up into 4 equally sized sections covering approximately 5 minutes of blimp flight each. The flight data consists of the blimp exhibiting standard blimp maneuvers. Slow and fast forward motion and turns are captured, as well as starts and stops which exercises the main thrusters' full range of motion. The tracking algorithms were tested on each section with the remaining sections used for learning of the GP models and parameters for the physics-based models. The GP models are learned with approximately 900 points by subsampling of the full data set. For the prediction model, we use every third point from the full training data. The particle filters were run with 2000 particles. Hyperparameter estimation for the all GP models is done offline, and the term $K^{-1}\mathbf{y}$ precomputed and cached. All other computations are done online under real time conditions.

**Table 8** Tracking mean log likelihood (MLL, higher values are better)

| | *Param* | *GP* | *EGP* | *hetGP* | *sparseGP* |
|---|---|---|---|---|---|
| *UKF* | 10.1 ± 0.6 | 14.9 ± 0.5 | 16.2 ± 0.3 | 16.9 ± 0.3 | 14.3 ± 0.6 |
| *EKF* | 8.4 ± 1.0 | 13.0 ± 0.2 | 14.4 ± 0.9 | 15.1 ± 0.8 | 12.8 ± 0.6 |
| *PF* | –4.5 ± 4.2 | 9.4 ± 1.9 | 10.7 ± 2.1 | na | 9.2 ± 1.8 |

**Table 9** Run time (s) per filter iteration

| | *Param* | *GP* | *EGP* | *hetGP* | *sparseGP* |
|---|---|---|---|---|---|
| *UKF* | 0.33 ± 0.1 | 1.28 ± 0.3 | 1.45 ± 0.2 | 2.02 ± 0.34 | 0.48 ± 0.11 |
| *EKF* | 0.21 ± 0.1 | 0.29 ± 0.1 | 0.53 ± 0.2 | 0.66 ± 0.15 | 0.06 ± 0.03 |
| *PF* | 31 ± 6 | 449 ± 21 | 492 ± 34 | na | 106 ± 16 |

The results are broken up into three charts. First, Table 7 shows the percentage improvement of tracking quality with various combinations of Bayes filters and models over the baseline filters using parametric models (*hetGP* and *sparseGP* will be discussed in Sect. 6.3.2). Specifically, the percentage improvement of RMS tracking error in position, rotation, velocity, and rotational velocity are averaged to get an aggregate improvement. The results show that GP-BayesFilters are clearly superior to their parametric Bayes filter counterparts. Enhanced-GP filters are consistently better than filters using zero-mean GP models (third column *vs*. second column), and UKF versions of all filters outperform their EKF counterparts (first row *vs*. second row).

Table 8 shows the mean log likelihood of the ground truth state given the estimated state distribution. The closer the estimated state is to the ground truth state, and the smaller the uncertainty of the state estimate, the higher the log likelihood. This table can be used to compare the tracking performance across different filter algorithms. The UKF has the best performance. It improves when using the GP prediction and observation models, and is even better when using enhanced-GP models. Particle filter tracking performance is surprisingly poor, this is likely caused by an insufficient number of particles for the size of the state space and the tracking uncertainty. Also, the distributions of the particles are converted into a Gaussian in order to compute the log likelihood.

Finally, Table 9 shows the run-time per iteration of the filter. The GP-EKF is more than four times faster than the GP-UKF, and GP-PFs have prohibitively large computation times for this application. Notice that GP-PF run-times are proportionately much slower than their parametric counterpart. Computation for GP-PF is much more extensive since prediction uncertainty needs to be calculated for each point per step, whereas the parametric PF uses a static uncertainty.

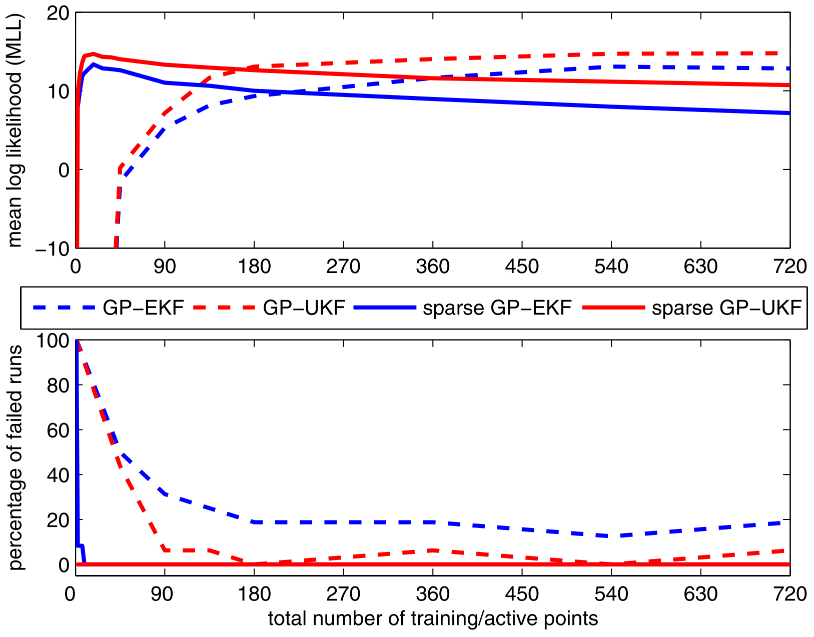

**Fig. 3** Tracking performance for different numbers of training points/active points. *Dashed lines* provide results when randomly choosing subsets of training points, *solid lines* give results when using sparse GP techniques. GP-UKF shows better accuracy and robustness compared to GP-EKF as the number of training points for the GP is reduced for both normal and sparse GPs. Sparse GP-UKF and GP-EKF show very high accuracy even as the number of active points approaches zero

### *6.3.1 Training data sparsity*

The next experiment tests how the tracking accuracy of GP-EKFs and GP-UKFs changes when only smaller amounts of training data are available. For each level of data sparseness, we perform cross validation by selecting 16 sets containing the corresponding number of training points. These sets were then tested on hold out data, just as in the four-fold cross validation structure from the previous experiment. These experiments are evaluated using two measures, as shown in Fig. 3 (sparse GP results will be discussed in Sect. 6.3.2). The first measure is the percentage of runs that failed to accurately track the blimp. A run is considered a failure if the ground truth state goes beyond three sigma bounds of the estimated state. In such cases, the filter has completely lost track of the blimp and is unlikely to recover. The other measure is the MLL of the remaining successful runs. As illustrated by the dashed lines in the figure, there is a fairly smooth degradation of accuracy for both methods. The tracking only experiences drastic reduction in accuracy with less than about 130 training points. This result also shows GP-UKF to be more accurate and more robust than GP-EKF at all levels of training data.

In an additional experiment we investigate how the GP-UKF technique behaves when the system transits through a section of the state space that was not observed during training. To do so, we removed data for specific motor commands, in this case left turns with the tail motor, from the GP training set. This reduced GP is then used by a GP-UKF for tracking. The results, available in Fig. 4, show the ground truth data along with the GP-UKF state estimate for both full and reduced training data sets. The blue (gray) lines indicate the three-$\sigma$ uncertainty bounds. The shaded region indicates the frames in which the tail motor was pushing left. The uncertainty of the tracking prediction increases precisely in the region where there is less training data. This covariance increase keeps the ground truth within three-$\sigma$ of the mean estimate even though the mean error increases. Note that the uncertainty of the reduced data GP-UKF reverts to that of the full data GP-UKF soon after this test period.

### *6.3.2 Heteroscedastic and sparse GPs*

The next experiments test the use of sparse and heteroscedastic GP models for blimp tracking. As stated earlier, GP-BayesFilters can naturally incorporate modified or sparse GPs in order to improve accuracy or increase efficiency.

We first test heteroscedastic GP models. As noted in Sect. 3.1.4, standard GPs assume that the noise of the process is the same for all states. Different uncertainties are only due to different data sparsity. Heteroscedastic GPs overcome this limitation by learning state dependent noise models. The tracking results for GP-BayesFilters with heteroscedastic GPs (*hetGP*) are shown in Tables 7 through 9. As can be seen, the tracking quality is consistently better than with regular GPs. In fact, *hetGP* obtains the best results of the different GP models examined. However there is a speed penalty due to the increased complexity of heteroscedastic GPs. This current implementation of *hetGP* is about twice as slow as the regular *GP* models.

Figure 5 illustrates the key difference between heteroscedastic and regular GPs during tracking. The shaded regions indicate the times in which the tail motor is active.

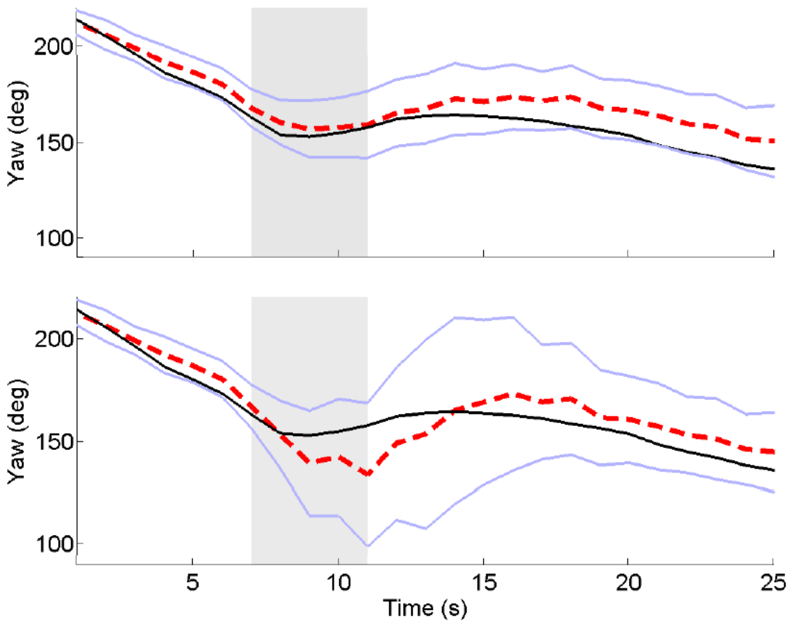

**Fig. 4** The figure shows the effect of eliminating training data associated with a left turn. Along the $x$-axis, the top plot shows the temporal evolution of ground truth yaw (*black line*) along with the GP-UKF yaw estimate (*red dashed line*) using all available training data. The tail motor is active within the *shaded region*. The *bottom plot* shows the same command sequence with all left turn training data eliminated. Notice how the GP-UKF automatically increases the estimate error covariance (*blue/gray lines*) due to increased model uncertainty (due to limited training data, see also Fig. 1)

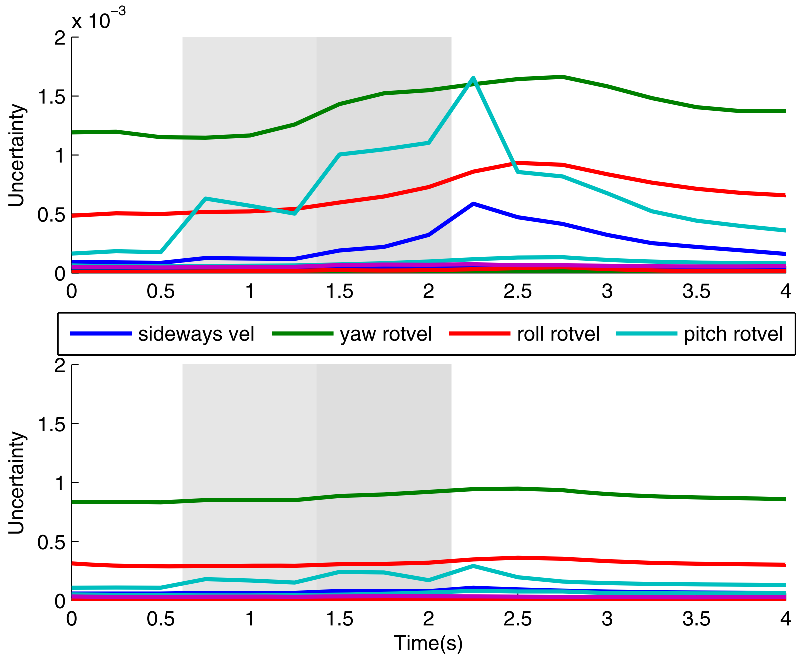

**Fig. 5** Tracking uncertainties of different state components when using heteroscedastic GP (*top*) and regular GP (*bottom*). The intensity of shading indicates tail motor power. Heteroscedastic GP has higher and more variable uncertainty compared to regular GP when the tail motor is active (*shaded region*)

The upper plot shows uncertainties in different state components when using heteroscedastic GPs and the lower shows uncertainties with regular GPs. As can be seen, the uncertainty of the heteroscedastic GP tracker increases whenever the tail motor transitions between different power levels. The uncertainty is also correlated with the overall yaw velocity of the blimp, since the yaw velocity increases the longer the motor is active. The faster the blimp is rotating, the more uncertain different dimensions of the state estimates become. The uncertainties that are most affected are the ones governing sideways velocity, and the rotational velocities of the blimp. This makes sense since the tail motor also causes a sideways movement as well as rotation. These changes in uncertainty are not seen when using regular GPs. This shows that heteroscedastic GPs do a better job at modeling the uncertainty in the blimp system.

Even when large amounts of training data are available, it might be advantageous to reduce the number of GP points in order to increase efficiency of the GP-BayesFilter. As demonstrated in Sect. 6.3.1, the quality of the filter decreases when subsets of training points are chosen randomly. Here we demonstrate that sparse GP techniques can be applied to significantly speed up GP-BayesFilters without sacrificing accuracy. Tracking results for GP-BayesFilters with the sparse GP technique introduced by Snelson and Ghahramani (2006) are shown in the last column of Tables 7 through 9. These results are based on 50 active points. The sparse GP-BayesFilters show similar accuracy to regular GP-BayesFilters, however, with much lower run-times.

The next experiment is related to the earlier data sparsity experiment with the results shown in Fig. 3. Instead of varying the number of training points, we vary the number of active points used by the sparse GP. The sparse GP-UKF and GP-EKF show very high accuracy even when the number of active points are greatly reduced (solid lines). There are two interesting observations in this result. First, the tracking accuracy decreases slightly as the number of active points used increases. As the number of active points increases, the number of parameters in the optimization increases. We hypothesize that the GPs with more active points need even more optimization iterations than we used in our experiments. Right now, the number of optimization iterations for all the sparse GPs are fixed. Another observation is that although the sparse GP seem to have lower MLLs than the regular GPs when using more data, this may not be the case since failed runs from the regular GPs are excluded from the MLL calculation.

## 6.4 Synthetic experiment

The previous experiment showed that GP-UKFs and GP-EKFs have very similar behavior. Here, we demonstrate the somewhat surprising result that GP-EKFs can handle certain sparseness conditions better than GP-UKFs.

The experimental setup is illustrated in Fig. 6. The filters use range observations to known landmarks to track a robot moving in a square circuit. The robot uses the following motion model:

$$p(t+1) = p(t) + v(t) + \epsilon_p, \tag{31}$$

$$v(t+1) = v(t) + u(t) + \epsilon_v, \tag{32}$$

where $p$ denotes position, $v$ velocity of the robot, and $u$ the control input. $\epsilon_p$ and $\epsilon_v$ are zero mean Gaussian noise for position and velocity, respectively. The robot receives as an observation the range to one of the landmarks. Observations are made for each landmark in a cyclic fashion.

The training data for the GPs is obtained from a typical run of the robot. The GPs for the motion model learn the change in position and velocity from one time step to the next. This is similar to the modeling of the blimp dynamics. The observation model uses as training data range information and positions from the robot during the training run. That is, observation training data only comes from locations visited during the training run.

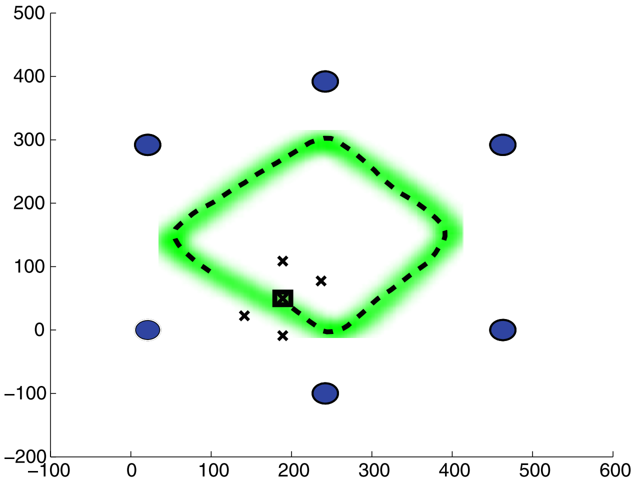

**Fig. 6** The robot is indicated by the *square* and the landmarks by the *circles*. The *dashed line* represents the trajectory of the robot. The (*green/grey*) *shaded region* indicates the density of training data. The *five crosses* indicate sigma points of the GP-UKF during a tracking run. Notice that the four outer points are in areas of low training data density

**Table 10** Tracking error for synthetic test

| | position | velocity |
|---|---|---|
| *GP-UKF* | 74.6 ± 0.34 | 8.0 ± 0.003 |
| *GP-EKF* | 48.2 ± 0.08 | 7.2 ± 0.005 |
| *GP-PF* | 43.8 ± 0.07 | 6.0 ± 0.003 |

As can be seen in Table 10, the tracking performance of GP-UKF is significantly worse than that of GP-EKF and GP-PF. This is a result of the sigma points being spread out around the mean prediction of the filter. In this experiment, however, the training data only covers a small envelope around the robot trajectory. Therefore, it can happen that sigma points are outside the training data and thus receive poor GP estimates. The GP-EKF does not suffer from this problem since all GP predictions are made from the mean point which is well within the coverage of the training data. GP-PFs handle such problems by assigning very low weights to samples with inaccurate GP models.

This experiment indicates that one must be careful when using GP-UKFs to select training points that ensure broad enough coverage of the operational space of the system. Another way to alleviate this problem could be to use different parameters for the UKF. These UKF parameters specify the weights given to each sigma point when reconstructing the Gaussian. In this work we used UKF parameter values of $\alpha = 1e^{-3}$, $\beta = 2$, and $\kappa = 0$. These values are optimal if the underlying distribution is indeed Gaussian, however different values can be used to force the weight of the central/mean sigma point to be higher. This will give the outer sigma points with low training data density lower weights thereby achieving higher propagation accuracy.

## 7 Conclusions and future work

Recently, several researchers have demonstrated that Gaussian process regression models are well suited as components of Bayesian filters. GPs are non-parametric models that can be learned from training data and that provide estimates that take both sensor noise and model uncertainty due to data sparsity into account. In this paper, we introduced GP-BayesFilters, the integration of Gaussian process prediction and observation models into generic Bayesian filtering techniques. Specifically, we developed GP-PFs, which combine GP models with particle filtering; GP-UKFs, which incorporate GP models into unscented Kalman filters; and we showed how GP models can be linearized and incorporated into extended Kalman filters. In addition to developing the algorithms, we provide a complexity analysis of the different instances of GP-BayesFilters.

In our experiments, all versions of GP-BayesFilters outperform their parametric counterparts. Typically, GP-UKFs perform slightly superior to GP-EKFs, at the cost of higher computational complexity. However, one experiment demonstrates that GP-EKFs can outperform GP-UKFs when training data is sparse and thus does not cover all sigma points generated during tracking. We additionally demonstrate how the accuracy and efficiency of GP-BayesFilters can be increased by applying heteroscedastic and sparse GP techniques. The combination with parametric models can result in additional improvements by reducing the need for extensive training data.

In general, however, the increased accuracy of GP-BayesFilters comes at the cost of increased computational complexity and the need for training data that covers the complete operational space of the dynamical system. We thus conjecture that GP-BayesFilters are most useful when high accuracy is needed or for difficult to model dynamical systems. Furthermore, an enhanced GP should be used whenever a reasonable, parametric model is available.

A further disadvantage of GP-BayesFilters is the need for accurate ground truth data for learning the GP models. In many applications, ground truth data is either unavailable, or can only be determined approximately. GP latent variable models (Lawrence 2004; Wang et al. 2006) were developed to handle cases in which no ground truth is available for GP input values. They are thus well suited for cases with uncertain training data, and their extension to learning GP-BayesFilters has recently been investigated with extremely promising results (Ko and Fox 2009). Finally, the predictive uncertainty of the GPs used in this paper assumed diagonal error covariance matrices. An interesting direction for future work is learning fully correlated matrices.

**Acknowledgements** We would like to thank Dirk Hähnel and Daniel Klein for their help in developing the initial GP-UKF framework, the blimp hardware, and the parametric model for blimp dynamics. This work was supported in part by DARPA's ASSIST Program (contract number NBCH-C-05-0137), by the NSF under contract number IIS-0812671, and by ONR MURI grant number N00014-07-1-0749.

## References

Abbeel, P., Coates, A., Montemerlo, M., Ng, A., & Thrun, S. (2005). Discriminative training of Kalman filters. In *Proceedings of robotics: science and systems (RSS)*.

Bar-Shalom, Y., Li, X.-R., & Kirubarajan, T. (2001). *Estimation with applications to tracking and navigation*. New York: Wiley.

Brooks, A., Makarenko, A., & Upcroft, B. (2006). Gaussian process models for sensor-centric robot localisation. In *Proceedings of the IEEE international conference on robotics & automation (ICRA)* (pp. 56–61).

Csató, L., & Opper, M. (2002). Sparse on-line Gaussian processes. *Neural Computation*, *14*(3), 641–668.

Deshpande, A. D., Ko, J., & Matsuoka, Y. (2009). Anatomically correct testbed hand control: muscle and joint control strategies. In *Proceedings of the IEEE international conference on robotics & automation (ICRA)*.

Doucet, A., de Freitas, J. F. G., Murphy, K., & Russell, S. (2000). Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proceedings of the conference on uncertainty in artificial intelligence (UAI)*.

Edakunni, N., Schaal, S., & Vijayakumar, S. (2007). Kernel carpentry for online regression using randomly varying coefficient model. In *Proceedings of the international joint conference on artificial intelligence (IJCAI)*.

Engel, Y., Szabo, P., & Volkinshtein, D. (2006). Learning to control an octopus arm with Gaussian process temporal difference methods. In Y. Weiss, B. Schölkopf, & J. Platt (Eds.), *Advances in neural information processing systems 18 (NIPS)* (pp. 347–354). Cambridge: MIT Press.

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

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

Girard, A., Rasmussen, C., Quiñonero Candela, J., & Murray-Smith, R. (2005). Gaussian process priors with uncertain inputs—application to multiple-step ahead time series forecasting. In *Advances in neural information processing systems 15 (NIPS)*. Cambridge: MIT Press.

Goldberg, P., Williams, C. K. I., & Bishop, C. (1998). Regression with input-dependent noise: A Gaussian process treatment. In *Advances in neural information processing systems 10 (NIPS)* (pp. 493–499). Cambridge: MIT Press.

Gomes, S. B. V., & Ramos, J. G. Jr. (1998). Airship dynamic modeling for autonomous operation. *Proceedings of the IEEE international conference on robotics & automation (ICRA)* (p. 4).

Grimes, D., Chalodhorn, R., & Rao, R. (2006). Dynamic imitation in a humanoid robot through nonparametric probabilistic inference. In *Proceedings of robotics: science and systems (RSS)*.

Julier, S., & Uhlmann, J. (1997). A new extension of the Kalman filter to nonlinear systems. In *International symposium on aerospace/defense sensing, simulation and controls*.

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

Ko, J., & Fox, D. (2008). GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. In *Proceedings 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 *Proceedings of robotics: science and systems (RSS)*.

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

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

Kocijan, J., Girard, A., Banko, B., & Murray-Smith, R. (2003). Dynamic systems identification with Gaussian processes. In *Proceedings of 4th IMACS symposium on mathematical modelling (MathMod)* (pp. 776–784).

Lawrence, N. (2004). Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in neural information processing systems 17 (NIPS)* (p. 2004).

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

Lawrence, N. D. http://www.dcs.shef.ac.uk/~neil/fgplvm/.

Limketkai, B., Fox, D., & Liao, L. (2007). CRF-filters: discriminative particle filters for sequential state estimation. In *Proceedings of the IEEE international conference on robotics & automation (ICRA)*.

Liu, K., Hertzmann, A., & Popovic, Z. (2005). Learning physics-based motion style with nonlinear inverse optimization. In *ACM transactions on graphics (Proceedings of SIGGRAPH)*.

Nguyen, D., & Peters, J. (2008). Local Gaussian processes regression for real-time model-based robot control. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)*.

Nott, D. (2006). Semiparametric estimation of mean and variance functions for non-Gaussian data. *Computational Statistics*, *21*(3–4), 603–620.

Paciorek, C. (2003). Nonstationary Gaussian processes for regression and spatial modelling.

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

Plagemann, C., Kersting, K., & Burgard, W. (2008). Nonstationary Gaussian process regression using point estimates of local smoothness. In *ECML PKDD '08: proceedings of the European conference on machine learning and knowledge discovery in databases, part II* (pp. 204–219), Antwerp, Belgium. Berlin: Springer. ISBN:978-3-540-87480-5, doi:10.1007/978-3-540-87481-2_14

Quinonero-Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, *6*, 1939–1959.

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

Sanner, R., & Slotine, J. (1991). Stable adaptive control and recursive identification using radial Gaussian networks. In *Proceedings of IEEE conference on decision and control (CDC)*.

Schölkopf, B., Smola, A., Williamson, R., & Bartlett, P. (2000). Letter communicated by John Platt new support vector algorithms.

Seeger, M., & Williams, C. (2003). Fast forward selection to speed up sparse Gaussian process regression. In *Workshop on AI and Statistics 9*.

Seo, S., Wallat, M., Graepel, T., & Obermayer, K. (2000). Gaussian process regression: Active data selection and test point rejection. In *German association for pattern recognition (DAGM) symposium* (pp. 27–34).

Smola, A., & Bartlett, P. (2001). Sparse greedy Gaussian process regression. In *Advances in neural information processing systems (NIPS)* (pp. 619–625). Cambridge: MIT Press.

Snelson, E., & Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In *Advances in neural information processing systems (NIPS)* (pp. 1257–1264). Cambridge: MIT Press.

Stevens, B. L., & Lewis, F. L. (1992). *Aircraft control and simulation*. New York: Wiley.

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

Wang, J., Fleet, D., & Hertzmann, A. (2006). Gaussian process dynamical models. In *Advances in neural information processing systems 18 (NIPS)* (pp. 1441–1448). Cambridge: MIT Press.

**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 and Director of the Robotics and State Estimation Lab in the Computer Science & Engineering Department at the University of Washington, 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. Along with his colleagues, he introduced particle filters as a powerful tool for state estimation in robotics. 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.