GP-BayesFilters: Bayesian Filtering Using Gaussian Process Prediction and Observation Models

Jonathan Ko and Dieter Fox

Dept. of Computer Science & Engineering,
University of Washington,
Seattle, WA

***Abstract*— Bayesian filtering is a general framework for recursively estimating the state of a dynamical system. The most common instantiations of Bayes filters are Kalman filters (extended and unscented) and particle filters. Key components of each Bayes filter are probabilistic prediction and observation models. Recently, Gaussian processes have been introduced as a non-parametric technique for learning such models from training data. In the context of unscented Kalman filters, these models have been shown to provide estimates that can be superior to those achieved with standard, parametric models. In this paper we show how Gaussian process models can be integrated into other Bayes filters, namely particle filters and extended Kalman filters. We provide a complexity analysis of these filters and evaluate the alternative techniques using data collected with an autonomous micro-blimp.**

I. Introduction

Estimating the state of a dynamical 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 [13]. 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. Typically, these models are parametric descriptions of the involved processes [13]. However, parametric models are not always able to capture all aspects of a dynamical system.

To overcome the limitations of parametric models, researchers have recently introduced non-parametric, Gaussian process (GP) regression models [12] to learn prediction and observation models for dynamical systems. GPs have been applied successfully to the problem of learning predictive state models [3, 4, 9]. The fact that GP regression models provide uncertainty estimates for their predictions allows them to be readily incorporated into particle filters as observation models [2] or as improved sampling distributions [10]. Ko and colleagues introduced GP-UKFs, which combine GP prediction and observation models with unscented Kalman filters. Using data collected with a robotic blimp they demonstrated that GP-UKFs outperform parametric unscented Kalman filters and that the performance of GP-UKFs can be increased by combining GP models with parametric models [7].

In this paper we investigate the integration of Gaussian Processes (GP) into different forms of Bayes filters. In addition to GP-UKFs, the previously introduced combination with unscented Kalman filters, we show how GP prediction and observation models can be combined with particle filters (GP-PF) and extended Kalman filters (GP-EKF). The development of GP-EKFs requires a linearization of GP regression models, which we derive in this paper. We furthermore 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 providing the necessary background on Bayesian filtering and Gaussian processes, we introduce the different instantiations of GP-BayesFilters in Section III. Section IV presents the experimental evaluation. We conclude in Section V.

II. Background of GP-BayesFilters

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

A. 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:

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

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 $p(\mathbf{x}_k \mid \mathbf{x}_{k-1},\mathbf{u}_{k-1})$ is the *prediction model*, a probabilistic model of the system dynamics. $p(\mathbf{z}_k \mid \mathbf{x}_k)$, 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 [13] for several examples. In GP-BayesFilters, both prediction and observation models are learned from training data using non-parametric, Gaussian process regression.

B. Gaussian Processes for Regression

Gaussian processes (GP) 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 [12].

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

$$y_i = f(\mathbf{x}_i) + \epsilon \ , \tag{2}$$

where ε is zero mean, additive Gaussian noise with variance σ_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

$$\mathrm{GP}_\mu \left(\mathbf{x}_*, D\right) = \mathbf{k}_*^T [K + \sigma_n^2 I]^{-1} \mathbf{y} \tag{3}$$

and variance

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

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 GP_Σ, 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:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}')W(\mathbf{x}-\mathbf{x}')^T} , \tag{5}$$

where σ_f^2 is the signal variance. The diagonal matrix W defines the smoothness of the process along the different input dimensions.

The GP parameters $\boldsymbol{\theta} = [W, \sigma_f, \sigma_n]$, describing the kernel function (5) and the process noise (2), respectively, are called the hyperparameters of the Gaussian process. These hyperparameters can be learned by maximizing the log likelihood of the training data using numerical optimization techniques such as conjugate gradient descent [12].

C. 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 (1). The training data for each GP is 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

$$D_p = \langle (X, U), X' \rangle \tag{6}$$

$$D_o = \langle X, Z \rangle \ , \tag{7}$$

where X is a matrix containing ground truth states, and $X' = [\Delta x_1, \Delta x_2, ..., \Delta x_k]$ 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

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) \approx$$
$$\mathcal{N}\left(\mathrm{GP}_\mu([\mathbf{x}_{k-1}, \mathbf{u}_{k-1}], D_p), \mathrm{GP}_\Sigma([\mathbf{x}_{k-1}, \mathbf{u}_{k-1}], D_p)\right) \tag{8}$$

and

$$p(\mathbf{z}_k|\mathbf{x}_k) \quad \approx \quad \mathcal{N}\left(\mathrm{GP}_\mu(\mathbf{x}_k, D_o), \mathrm{GP}_\Sigma(\mathbf{x}_k, D_o)\right) , \tag{9}$$

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 Section III.

GPs are typically defined for scalar outputs, and GP-BayesFilters represent models for vectorial outputs by learning a different GP for each output dimension. As a result, the noise covariances GP_Σ are diagonal matrices. Another issue to consider is that GPs assume a zero mean prior over the outputs of the functions. A direct ramification of this assumption is that the GP predictions tend towards zero as the distance between the test input and the training data increases. In practice, this problem can be reduced by collecting sufficient training data covering possible states, controls, and observations, and by incorporating parametric models into the GP, as shown in [7].

III. Instantiations of GP-BayesFilters

We will now show how GP models can be incorporated into different instantiations of Bayes filters. Specifically, we present algorithms for GP integration into particle filters, extended Kalman filters, and unscented Kalman filters. For notation, we will stick close to the versions presented in [13].

A. GP-PF: Gaussian Process Particle Filters

Particle filters are sample-based implementations of Bayes filters. The key idea of particle filters is to 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 [13]. Table I 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 (8). Here, $\mathrm{GP}([\mathbf{x}_{k-1}^m, \mathbf{u}_{k-1}], D_p)$ is short for the Gaussian represented by $\mathcal{N}\left(\mathrm{GP}_\mu([\mathbf{x}_{k-1}^m, \mathbf{u}_{k-1}], D_p), \mathrm{GP}_\Sigma([\mathbf{x}_{k-1}^m, \mathbf{u}_{k-1}], D_p)\right)$. 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 most recent measurement $\mathbf{z}_k$ given the sampled state $\mathbf{x}_k^m$. This likelihood can be easily extracted from the GP observation model defined in (9). All other steps are identical to the generic particle filter

```
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 $\mathbf{x}_k^m \sim \mathrm{GP}([\mathbf{x}_{k-1}^m, \mathbf{u}_{k-1}], D_p)$
5:      $w_k^{[m]} = \mathcal{N}\left(\mathbf{z}_k; \mathrm{GP}_\mu(\mathbf{x}_k^m, D_o), \mathrm{GP}_\Sigma(\mathbf{x}_k^m, D_o)\right)$
6:      $\hat{\mathcal{X}}_k = \hat{\mathcal{X}}_k + \langle \mathbf{x}_k^m, w_k^{[m]} \rangle$
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 I
THE GP-PF ALGORITHM.

```
1:  Algorithm GP-EKF($\mu_{k-1}, \Sigma_{k-1}, \mathbf{u}_{k-1}, \mathbf{z}_k$):
2:  $\hat{\mu}_k = \mathrm{GP}_\mu([\mu_{k-1}, \mathbf{u}_{k-1}], D_p)$
3:  $Q_k = \mathrm{GP}_\Sigma([\mu_{k-1}, \mathbf{u}_{k-1}], D_p)$
4:  $G_k = \frac{\partial \mathrm{GP}_\mu([\mu_{k-1}, \mathbf{u}_{k-1}], D_p)}{\partial \mathbf{x}_{k-1}}$
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:  $R_k = \mathrm{GP}_\Sigma(\hat{\mu}_k, D_o)$
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 $\mu_k, \Sigma_k$
```

TABLE II
THE GP-EKF ALGORITHM.

algorithm, where Steps 8 through 11 implement the resampling step (see [13]).

B. GP-EKF: Gaussian Process Extended Kalman Filters

In addition to the GP mean and covariance estimates used in the GP-PF, the incorporation of GP models into the extended Kalman filter requires a linearization of the GP function. Fortunately, this linearization follows directly from the definition of (3). For each output dimension, the derivative of the GP mean function is:

$$\frac{\partial\left(\mathrm{GP}_\mu\left(\mathbf{x}_*, D\right)\right)}{\partial(\mathbf{x}_*)} = \frac{\partial(\mathbf{k}_*)}{\partial(\mathbf{x}_*)}^T \left[K + \delta_n^2 I\right]^{-1} \mathbf{y}. \tag{10}$$

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

$$\frac{\partial(\mathbf{k}_*)}{\partial(\mathbf{x}_*)} = \begin{bmatrix} \frac{\partial(k(\mathbf{x}_*, \mathbf{x_1}))}{\partial(\mathbf{x}_*[1])} & \cdots & \frac{\partial(k(\mathbf{x}_*, \mathbf{x_1}))}{\partial(\mathbf{x}_*[d])} \\ \vdots & \ddots & \vdots \\ \frac{\partial(k(\mathbf{x}_*, \mathbf{x_n}))}{\partial \mathbf{x}_*[1])} & \cdots & \frac{\partial(k(\mathbf{x}_*, \mathbf{x_n}))}{\mathbf{x}_*[d])} \end{bmatrix}, \tag{11}$$

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

$$\frac{\partial(k(\mathbf{x}_*, \mathbf{x}))}{\partial(\mathbf{x}_*[i])} = -W_{ii}\sigma_f^2(\mathbf{x}_*[i] - \mathbf{x}[i])e^{-\frac{1}{2}(\mathbf{x}_* - \mathbf{x})W(\mathbf{x}_* - \mathbf{x})^T}. \tag{12}$$

(10) defines the d-dimensional Jacobian vector for a single output dimension. The full lxd Jacobian of a prediction or observation model is determined by stacking l 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 II. Step 2 uses the GP prediction model (3) 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 extracted from the GP via (10). 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.

C. GP-UKF: Gaussian Process Unscented Kalman Filters

The GP-UKF algorithm in Table III is restated for completeness of exposition [7]. 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. Steps 5 and 6 are identical to the standard unscented 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 [13]). 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.

D. GP-BayesFilter Complexity Analysis

This section examines the runtime complexity of the different algorithms. In typical tracking applications, the number

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

TABLE III
THE GP-UKF ALGORITHM.

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).

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

$$C_{\mathrm{pf}} \quad = \quad M\ (C_\mu + C_\Sigma). \tag{13}$$

The GP-EKF algorithm requires one GP mean and variance computation plus one GP Taylor series expansion step:

$$C_{\mathrm{ekf}} \quad = \quad C_\mu + C_\Sigma + C_{\mathrm{tse}} \tag{14}$$

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

$$C_{\mathrm{ukf}} \quad = \quad (2d+1)C_\mu + C_\Sigma. \tag{15}$$

To get a more accurate measure, we have to consider the complexity of the GP operations. The core of each GP mean prediction operation consists of a kernel evaluation (5) followed by a multiplication (we treat (12) as computationally equivalent to a kernel evaluation). We denote these costs as C_{kern} and C_{mult}, respectively. The computation of the mean function $\mathrm{GP}_\mu(\mathbf{x}_*, D)$ defined in (3) 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 to model each output dimension. We assume the number of output dimensions is equal to the dimensionality of the state space:

$$C_\mu \quad = \quad nd\,(C_{\mathrm{kern}} + C_{\mathrm{mult}}) \tag{16}$$

Note that the term $\left[K + \sigma_n^2 I\right]^{-1}\mathbf{y}$ in (3) and (10) is independent of $\mathbf{x}_*$ and can be cached. The computation of the covariance function $\mathrm{GP}_\Sigma(\mathbf{x}_*, D)$ given in (4) requires nd additional multiplications given proper caching:

$$C_\Sigma \quad = \quad nd\,C_{\mathrm{mult}}, \tag{17}$$

and similarly, the linearization of the GP, given in (12), also requires nd multiplications:

$$C_{\mathrm{tse}} \quad = \quad nd\,C_{\mathrm{mult}}. \tag{18}$$

The total costs in terms of kernel computations and multiplications are tabulated below.

TABLE IV
COMPUTATIONAL REQUIREMENTS

	C_{kern}	C_{mult}
GPUKF	$nd(2d+1)$	$nd(2d+2)$
GPEKF	nd	$3nd$
GPPF	Mnd	$2Mnd$

Unfortunately, there is no way to compare kernel operations directly to multiplications since various different kernels can be used. However, analysis in these terms can still be useful. First, the number of particles is the key determinant for the complexity of the GP-PF. As can be seen from Table V, GP-PF is not very competitive time-wise to the other filters. On the other hand, GP-PF is the only algorithm which can perform global localization. GP-UKF has roughly $2d$ times more kernel computations than GP-EKF. Since the kernel computation is generally much slower than a multiply operation, the cost for the kernel operations will dominate in these two algorithms.

IV. EXPERIMENTS

A. Robotic Blimp Experiments

1) Testbed: The experimental testbed for evaluating the GP-Bayes filters is a robotic micro-blimp. 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.

Observations for the filters come from two network cameras mounted in the laboratory. Observations are formed by

TABLE V

BLIMP TRACKING QUALITY

Tracking algorithm	MLL	$\mathbf{p}$(mm)	ξ(deg)	$\mathbf{v}$(mm/s)	ω(deg/s)	time(s)
$GPUKF$	14.9±0.5	89±1.3	4.7±0.2	50±0.4	4.5±0.1	1.28±0.3
$GPEKF$	13.0±0.2	93±1.4	5.2±0.1	52±0.5	4.6±0.1	0.29±0.1
$GPPF$	9.4±1.9	91±7.5	6.4±1.6	52±3.7	5.0±0.2	449.4±21
$paraUKF$	10.1±0.6	111±3.8	7.9±0.1	64±1.2	7.6±0.1	0.33±0.1
$paraEKF$	8.4±1.0	112±3.8	8.0±0.2	65±1.6	7.5±0.2	0.21±0.1
$paraPF$	-4.5±4.2	115±4.5	10.5±1.7	73±5.4	9.4±0.3	30.7±5.8

background subtraction followed by fitting an ellipse to the remaining pixels. The observations are then the parameters of the ellipse in image space.

A motion capture system is used to capture the ground truth positions of the blimp as it flies. The VICON motion capture (MOCAP) system tracks reflective markers attached to the blimp as 3D points in space. This data is used to train the GPs and parametric motion models, and to evaluate the tracking performance of the various filtering algorithms. The testbed software is written in a combination of C++ and MATLAB. Gaussian process code is from Neil Lawrence [8]. All experiments were performed on an Intel® Xeon™ running at 3.40GHz.

2) Parametric Prediction and Observation Models: The parametric motion model appropriate for the blimp was described in detail in [7]. The state of the blimp consists of position $\mathbf{p}$, orientation $\boldsymbol{\xi}$ parameterized by Euler angles, linear velocity $\mathbf{v}$, and angular velocity $\boldsymbol{\omega}$. The parametric motion model takes into account forces including gravity, buoyancy, thrust, and drag. This model is discretized in time in order to produce $\hat{g}$ which predicts the next state given the current state and control input.

The parametric observation model takes as input the six-dimensional pose of the blimp within the camera's coordinate system. It outputs the parameters of an ellipse in the camera's image space. The ellipse parameters are generated based on a shape model of the blimp.

3) Tracking Results: Experimental results 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 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 motion models were optimized using approximately 900 training points while the GP observation model used 800 points. The particle filters were run with 2000 particles.

Table V shows the tracking accuracy of the various filters. Tracking accuracy is shown separately for position, rotation, velocity, and rotational velocity. In addition, mean log likelihoods (MLL) of the ground truth given the state estimate and time per filter update are shown.

The three GP-Bayes filters have better accuracy in general than their parametric Bayes filter counterparts. GP-UKF outperforms GP-EKF by a small margin, however, the GP-EKF is more than four times faster than the GP-UKF. GP-PF has surprisingly poor accuracy. This is likely due to an insufficient number of particles for the size of the state space.

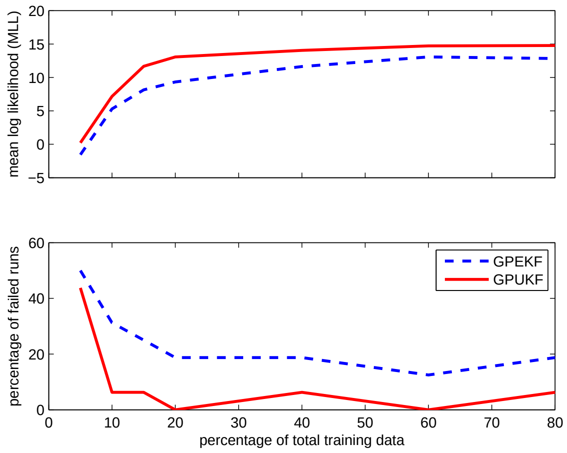

Fig. 1. GP-UKF shows better accuracy and robustness compared to GP-EKF as the number of training points for the GP is reduced.

4) Training Data Sparsity: The next experiment tests the tracking accuracy of the GP-EKF vs. GP-UKF as the number of training points is reduced. For each level of data sparseness, we perform cross validation by selecting 16 sets containing the corresponding number of training data. These sets were then tested on hold out data, just as in the four-fold cross validation structure from the previous experiment. Results for this experiment are shown using two measures. The first is the percentage of runs that failed to accurately track the blimp. A run is considered a failure if the MLL is less than -20, since such runs have completely lost track of the blimp. The other measure is the MLL of the remaining runs. There is a fairly smooth degradation of accuracy for both methods. The tracking only experiences drastic reduction in accuracy with less than 15% of the total training points (≈ 130). This result also shows GP-UKF to be more accurate and more robust than GP-EKF at all levels of training data.

B. Synthetic Experiment

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

The filters use range observations to known landmarks to track a robot moving in a square circuit. The robot uses the following motion model:

$$p(t+1) = p(t) + v(t) + \epsilon_p \tag{19}$$
$$v(t+1) = v(t) + u(t) + \epsilon_v, \tag{20}$$

where p denotes position, v velocity of the robot, and u the control input. ϵ_p and ϵ_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 training run of the robot. The GPs for the motion model learn the change in position and velocity from one time step to the

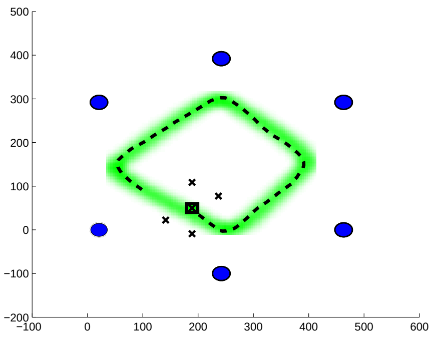

Fig. 2. The robot is indicated by the square and the landmarks by the circles. The dashed line represents the trajectory of the robot. The (green / grey) shaded region indicates the density of training data. The five crosses indicate sigma points of the GP-UKF during a tracking run. Notice that the four outer points are in areas of low training data density.

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.

TABLE VI

TRACKING ERROR FOR SYNTHETIC TEST

	$\mathbf{p}$	$\mathbf{v}$
$GPUKF$	74.6±0.34	8.0±0.003
$GPEKF$	48.2±0.08	7.2±0.005
$GPPF$	43.8±0.07	6.0±0.003

As can be seen in Table V, 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.

V. 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. In addition to the existing GP-UKF algorithm, we developed GP-PFs, which combine GP models with particle filtering. Furthermore, 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.

The additional operations needed for GP models can result in prohibitive complexity, especially in the case of particle filters. We thus conjecture that the best applications for GP-BayesFilters are those where computational complexity is not crucial, or where accuracy is very important. When efficiency is important, the use of *sparse* GP models can greatly reduce the computational complexity of GP regression [12]. Furthermore, as shown in [6], Gaussian process models can be combined with parametric models, resulting in further improved accuracy.

Acknowledgments

This work was supported in part by DARPA's ASSIST Program (contract numbers NBCH-C-05-0137).

References

[1] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo in Practice*. Springer-Verlag, New York, 2001.

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

[3] A. Girard, C. Rasmussen, J. Quiñonero Candela, and R. Murray-Smith. Gaussian process priors with uncertain inputs – application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems 15 (NIPS)*. 2005.

[4] D. Grimes, R. Chalodhorn, and R. Rao. Dynamic imitation in a humanoid robot through nonparametric probabilistic inference. In *Proc. of Robotics: Science and Systems*, 2006.

[5] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard. Most likely heteroscedastic gaussian process regression. In *Proc. of the International Conference on Machine Learning (ICML)*, 2007.

[6] 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.

[7] J. Ko, D.J. 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.

[8] N.D. Lawrence. http://www.dcs.shef.ac.uk/˜neil/fgplvm/.

[9] K. Liu, A. Hertzmann, and Z. Popovic. Learning physics-based motion style with nonlinear inverse optimization. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 2005.

[10] 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.

[11] V. Quoc, A. Smola, and S. Canu. Heteroscedastic Gaussian process regression. In *Proc. of the International Conference on Machine Learning (ICML)*, 2005.

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

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