Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp

Jonathan Ko* Daniel J. Klein† Dieter Fox* Dirk Haehnel‡

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

† *Dept. of Aeronautics & Astronautics, University of Washington, Seattle, WA*

‡ *Intel Research Seattle, Seattle, WA*

***Abstract*— Blimps are a promising platform for aerial robotics and have been studied extensively for this purpose. Unlike other aerial vehicles, blimps are relatively safe and also possess the ability to loiter for long periods. These advantages, however, have been difficult to exploit because blimp dynamics are complex and inherently non-linear. The classical approach to system modeling represents the system as an ordinary differential equation (ODE) based on Newtonian principles. A more recent modeling approach is based on representing state transitions as a Gaussian process (GP). In this paper, we present a general technique for system identification that combines these two modeling approaches into a single formulation. This is done by training a Gaussian process on the residual between the non-linear model and ground truth training data. The result is a *GP-enhanced* model that provides an estimate of uncertainty in addition to giving better state predictions than either ODE or GP alone. We show how the GP-enhanced model can be used in conjunction with reinforcement learning to generate a blimp controller that is superior to those learned with ODE or GP models alone.**

I. Introduction and Motivation

Unmanned aerial vehicles (UAVs) have become a helpful component for many applications where human operation is considered unnecessary or too dangerous. Blimps effectively combine the capabilities of airplanes with those of hot air balloons into one aircraft. This unique combination of maneuverability and the ability to float with relatively low power requirements makes a blimp an ideal research platform for sensor and control technology. Blimps have been studied in various contexts. So far, blimp controllers are mainly based on PID controllers [14], [15], [16] or non-linear dynamic models [1], [2], [3], [5].

System identification is the first step towards designing a controller for an autonomous blimp, and dynamical systems in general. A system model describes how the state changes from one instant to the next. The quality of a model can be measured by how well it predicts the next state given the current state and control input. A higher fidelity model results in improved state estimation and controller performance.

The result of classical dynamic modeling is an ordinary differential equation which describes the evolution of the state. The model can be formulated without collecting any training data, however extensive human knowledge is required. Another disadvantage of this approach is that the system noise is generally difficult to model. Gaussian process (GP) regression models have recently been applied to the problem of learning dynamic models from training data [4], [6]. GPs have several key properties that make them ideally suited to our problem. They are non-parametric, which lets them model a wide range of dynamical systems. Furthermore, they can automatically learn the smoothness and noise levels of the underlying system. Finally, they provide a notion of uncertainty about the learned process. This uncertainty can be very valuable when learning a controller.

Fig. 1. 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.

However, standard GPs assume that the process underlying the data is zero-mean, which is clearly not the case when learning a model of blimp dynamics. In order to overcome this problem, we combine dynamical and data-driven modeling to form a single *GP-enhanced* model. The GP-enhanced model begins with a classical non-linear dynamical model created by a human expert. The parameters of this blimp model are learned using ground truth data. Then a Gaussian process is used to model the residual between the prediction of the dynamical model and ground truth data. Experiments with an indoor blimp show that the GP-enhanced model outperforms both the classical non-linear approach and the pure GP-based approach. The GP-enhanced model is then used in reinforcement learning to learn a controller for the blimp.

This paper is organized as follows. The blimp hardware testbed used in the experiments is described in Section II. In Section III, the non-linear dynamics of the blimp are derived. Our approach to using Gaussian Processes for learning predictive models is described in Section IV. A blimp controller is built using reinforcement learning in Section V. Finally, experimental results illustrating the advantages of the GP-enhanced model are presented in Section VI.

II. Hardware Testbed

The blimp used in the experiments (see Fig. 1) is based on a commercial 5.5 foot (1.7 meter) blimp envelope [8] with a custom built gondola. The gondola includes a small XScale PXA271 based computer running Linux with Bluetooth, miniSD, and several sensors. The computer is connected to a servo controller, a motor controller, and a webcam. The motor controller board is used to control the speed of the tail motor in both the forward and reverse directions. The servo controller steers the two servos that rotate the gondola motors up to 360 degrees. The gondola motors, two ducted fans (GWS EDF-50), are controlled by two standard ESCs (Electronic Speed Controller) attached to the servo controller. The system is powered by two batteries. A 3.7V/200mAh lithium ion battery supplies the computer and the controllers. A bigger 7.4V/1200mAh lithium polymer battery drives the ducted fans, the tail motor, and the servos. The total weight of the blimp is about 350 grams. Neither the sensors nor the webcam were used in our experiments.

All experiments are carried out in a room equipped with a VICON motion capture system. Seven markers were attached in order to track the blimp in our test environment (see left image in Fig. 1). The mean measurement error of markers in the system is about $1cm$. The maximum capture rate of the system is $120Hz$, however, jitter on this system is rather high. The frame rate deviates from the ideal $120Hz$ with a standard deviation of $12Hz$. The system outputs positions of the markers only. These marker position values are converted to blimp states by the proper transformations. Velocities for the states are obtained by a smoothed calculation of the slope between sample points.

III. Nonlinear Blimp Dynamics

The non-linear deterministic model of the blimp derived in this section is based on standard dynamic and aeronautic principles. The objective is to create an autonomous non-linear plant model of the form,

$$\dot{\mathbf{s}}(t) = \mathbf{f}_{\boldsymbol{\gamma}}(\mathbf{s}(t), \mathbf{u}(t)), \tag{1}$$

were $\mathbf{s}(t)$ is the state, $\mathbf{u}(t)$ is the control, and $\boldsymbol{\gamma}$ is a vector of constants that includes the added mass and inertia, drag coefficients, center of mass, and volume. The state vector,

$$\mathbf{s} = \left[\mathbf{p}^T, \boldsymbol{\xi}^T, \mathbf{v}^T, \boldsymbol{\omega}^T\right]^T, \tag{2}$$

consists of position $\mathbf{p}^T = [x, y, z]$, orientation $\boldsymbol{\xi}^T = [\phi, \theta, \psi]$ parametrized by roll ϕ, pitch θ, and yaw ψ, translational velocity $\mathbf{v}^T = [U, V, W]$, and angular velocity $\boldsymbol{\omega}^T = [P, Q, R]$. A forward-right-down body-fixed reference frame is attached to the center of buoyancy of the blimp as shown in Fig. 2. The state is measured with respect to a traditional North-East-up inertial reference frame, however the velocity components are expressed in the body-fixed frame because the mass and inertia matrices are constant with respect to this frame. Note that these matrices are not necessarily diagonal due to added mass effects, which account for accelerating the surrounding air.

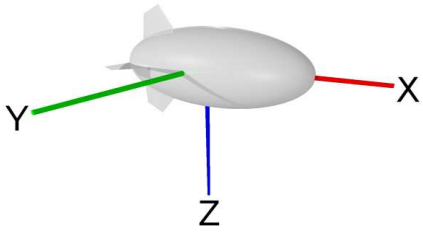

Fig. 2. Body fixed frame of reference for the blimp. Forwards is body X (red), right is body Y (green), and down is body Z (blue).

The gondola motors share a common shaft and thus always point at the same angle, μ. Further, the blimp is set up so that both gondola motors receive the same motor command. Appropriately, the blimp has a total of three control inputs,

$$\mathbf{u} = \left[u_t, u_g, u_\mu\right]^T, \tag{3}$$

which are the commands sent to each respective motor.

A detailed derivation of the rigid body dynamics of blimp-like vehicles can be found in [3], [13]. The resulting model has the form,

$$\dot{\mathbf{s}} = \frac{d}{dt}\begin{bmatrix}\mathbf{p}\\ \boldsymbol{\xi}\\ \mathbf{v}\\ \boldsymbol{\omega}\end{bmatrix} = \begin{bmatrix} R_b^e\mathbf{v} \\ H(\boldsymbol{\xi}) \\ M^{-1}\left(\sum \text{Forces} - \boldsymbol{\omega}\times M\mathbf{v}\right) \\ J^{-1}\left(\sum \text{Torques} - \boldsymbol{\omega}\times J\boldsymbol{\omega}\right)\end{bmatrix}. \tag{4}$$

Here, R_b^e is the rotation matrix from body-fixed to inertial reference, and H is the Euler kinematical matrix.

The sum of forces acting on the blimp system in (4) is evaluated in the body frame and consists of terms for each motor, gravity, buoyancy, and aerodynamics. The gravity and buoyancy forces can be computed directly,

$$\text{Force}_{gravity} = (R_b^e)^T[0, 0, -mg]^T \tag{5}$$

$$\text{Force}_{buoyancy} = (R_b^e)^T[0, 0, m\mathcal{V}\rho_{air}]^T, \tag{6}$$

where m and $\mathcal{V}$ are the mass and volume of the blimp, respectively, ρ_{air} is the density of air, and g is the gravitational constant. Following typical engineering practice, the remaining force terms are modeled parametrically:

$$\text{Force}_{aero\ drag} = -C_d\|\mathbf{v}\|\mathbf{v} \tag{7}$$

$$\text{Force}_{gondola\ motors} = f_g(u_g)[\cos(\mu), 0, -\sin(\mu)]^T \tag{8}$$

$$\text{Force}_{tail\ motor} = f_t(u_t)[0, -1, 0]^T. \tag{9}$$

A quadratic model was used for the mapping from the motor command to the resulting force for the gondola motors, f_g, and the tail motor, f_t.

Each of the above forces causes a torque about the center of buoyancy, about which the sum of torques is evaluated. Thus, each force term creates a corresponding torque term of the form $\text{Torque}_i = r_i \times \text{Force}_i$. Here r_i is a vector from the center of buoyancy to the application point of the i^{th} force. The only new term is the rotational drag, which is modeled as a pure moment,

$$\text{Torque}_{aero\ rot} = -C_r\boldsymbol{\omega}, \tag{10}$$

where C_r is a drag coefficient. The unknown parameter vector, $\boldsymbol{\gamma}$, includes all coefficients and vectors that cannot be measured directly. These parameters were learned by minimizing point-wise differences between simulated states and ground truth data from the motion capture system.

IV. Learning Predictive Models Using Gaussian Processes

A. Preliminaries

Gaussian processes (GP) are a powerful, non-parametric tool for regression in high dimensional spaces. Key advantages of GPs are their ability to provide uncertainty estimates and to learn the noise and smoothness parameters from training data. More information can be found in [12]. A GP can be thought of as a "Gaussian over functions". More precisely, a GP describes a stochastic process in which the random variables, in this case the outputs of the modeled function, are jointly Gaussian distributed. A Gaussian process is fully described by its mean and covariance functions.

The training set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$ is assumed to be drawn from the noisy process

$$y_i = f(\mathbf{x}_i) + \epsilon \ , \tag{11}$$

where $\mathbf{x}_i$ is an input vector in $\mathbb{R}^d$ and y_i is a scalar output in $\mathbb{R}$ (extension to multiple outputs is possible). The noise term ϵ is drawn from $\mathcal{N}(0, \sigma^2)$. For convenience, the inputs are aggregated into a matrix $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]$. The outputs are likewise aggregated, $\mathbf{y} = [y_1, y_2, ..., y_n]$. The joint distribution over the noisy outputs $\mathbf{y}$ given inputs X is a zero-mean Gaussian, and has the form,

$$p(\mathbf{y}|X) = \mathcal{N}(\mathbf{0}, K(X, X) + \sigma_n^2 I), \tag{12}$$

where $K(X, X)$ is the kernel matrix with elements $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The kernel function, $k(\mathbf{x}, \mathbf{x}')$, is a measure of the "closeness" between inputs. The term $\sigma_n^2 I$ introduces the Gaussian noise and plays a similar role to that of ϵ in (11).

The squared exponential is a commonly used kernel function and will be used in this paper. It is,

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 exp((-\frac{1}{2}(\mathbf{x} - \mathbf{x}')W(\mathbf{x} - \mathbf{x}')^T)), \tag{13}$$

where σ_f^2 is the signal variance. The diagonal matrix W contains the length scales for each input dimension. The value of W_{ii} is inversely proportional to the importance of the i-th input dimension. Learning the matrix W can thus be used for automatic relevance determination (ARD) [9].

Given a set of test inputs X_*, one would like to find the predictive outputs $\mathbf{f}_*$. The noisy training outputs $\mathbf{y}$ and the test output $\mathbf{f}_*$ are jointly Gaussian:

$$p(\mathbf{f}_*, \mathbf{y}|X_*, X) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X_*, X_*) & K(X_*, X) \\ K(X, X_*) & K(X, X) + \sigma_n^2 I \end{bmatrix}\right) \tag{14}$$

Since $\mathbf{y}$ is known, this Gaussian can be conditioned on $\mathbf{y}$ to obtain the predictive distribution for X_*

$$p(\mathbf{f}_*|\mathbf{y}, X_*, X) = \mathcal{N}(\boldsymbol{\mu}, \Sigma), \tag{15}$$

where

$$\boldsymbol{\mu} = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y} \tag{16}$$

$$\Sigma = K(X_*, X_*) - K(X_*, X)\left[K(X, X) + \sigma_n I\right]^{-1} K(X, X_*). \tag{17}$$

These equations show that the mean function is a linear combination of the training output $\mathbf{y}$, and the weight of each output is directly related to the correlation between X_* and the training input.

The parameters of the kernel function (13), $\boldsymbol{\theta} = [W, \sigma_f, \sigma_n]$, are called the hyperparameters of the Gaussian process. These hyperparameters can be learned by maximizing the log likelihood of the training outputs given the inputs,

$$\boldsymbol{\theta}_{max} = \underset{\theta}{\operatorname{argmax}} \left\{\log(p(\mathbf{y}|X, \boldsymbol{\theta}))\right\}. \tag{18}$$

The log term in (18) can be expressed as

$$\log(p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^T(K(X, X) + \sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log|K(X, X) + \sigma_n^2 I| - \frac{n}{2}\log 2\pi. \tag{19}$$

B. GP Modeling of Discrete Time Dynamic Processes

A discrete time dynamic process can be thought of as a series of states indexed by time and can be written as

$$\mathbf{s}(k+1) = \mathbf{s}(k) + \mathbf{g}(\mathbf{s}(k), \mathbf{u}(k)), \tag{20}$$

where k is the time index and $\mathbf{g}$ is a function which describes the dynamics of the system given the current state $\mathbf{s}$ and control inputs $\mathbf{u}$. A Gaussian process can be used to model this system by learning the function $\mathbf{g}$ based on training data consisting of a sequence of observed states and controls:

$$\mathbf{x}_k = [\mathbf{s}(k), \mathbf{u}(k)] \tag{21}$$

$$\mathbf{y}_k = \mathbf{g}(\mathbf{s}(k), \mathbf{u}(k)) \equiv \mathbf{s}(k+1) - \mathbf{s}(k). \tag{22}$$

As a results, the GP learns to predict the delta between two consecutive states conditioned on the previous state and the control input.

C. Sampling Trajectories from a Gaussian Process

Once the parameters of a GP are learned from training data, the GP can be used to simulate the evolution of the dynamic process. This is done by sequentially sampling states from the predictive distribution.

To see, assume that we have training data X and $\mathbf{y}$ generated according to (21) and (22), and an initial state and control input giving $\hat{\mathbf{x}}_0 = [\hat{\mathbf{s}}(0), \mathbf{u}(0)]$. The state at time 1 is then given by $\hat{\mathbf{s}}(1) = \hat{\mathbf{s}}(0) + \hat{\mathbf{y}}_0$, where $\hat{\mathbf{y}}_0$ is the GP prediction based on the training data and initial state. The predictive distribution $p(\hat{\mathbf{y}}_0|\mathbf{y}, \hat{\mathbf{x}}_0, X)$ is Gaussian with mean and variance given by (16) and (17), respectively (with X_* replaced by $\hat{\mathbf{x}}_0$). Sampling an instance of $\hat{\mathbf{y}}_0$ from this distribution provides the information needed to generate the state $\hat{\mathbf{s}}(1)$. A new control input $\mathbf{u}(1)$ along with $\hat{\mathbf{s}}(1)$ can then used to generate $\hat{\mathbf{s}}(2)$ in a similar way.

This process can be iterated until a complete trajectory is sampled. However, care has to be taken that the correlation between consecutive data points is considered. To do so, one needs to condition future points on the points sampled so far.

This can be done elegantly by adding already sampled points to the training data X and $\mathbf{y}$, thereby growing the kernel matrices and vectors used in the predictive distributions specified by (16) and (17). As a result, if enough trajectories are sampled from a particular state, the distribution of endpoints of these trajectories will properly represent the real distribution of endpoints.

D. Model Enhancement using the Deterministic Model

So far, the evolution of the dynamic system is represented by the Gaussian process alone. There are some drawbacks to this technique. Because there is a zero mean assumption in the GP, test states that are far away from the training states will have outputs that tend towards zero. This makes the choice of training data for the GP very important. However, the deterministic model developed in Section III has prediction quality which is independent of the location of the training data. The deterministic model can be combined with the GP model to give more accurate state predictions. The dynamic system equation then becomes,

$$\mathbf{s}(k+1) = \mathbf{s}(k) + \mathbf{f}(\mathbf{s}(k), \mathbf{u}(k)) + \mathbf{g}(\mathbf{s}(k), \mathbf{u}(k)), \quad (23)$$

where $\mathbf{f}$ describes the change in state given by the deterministic model. The function $\mathbf{g}$, which is modeled with a GP, is now only responsible for describing the residual between the ground truth data and the deterministic non-linear model. To generate training data for the GP, we again use a sequence of observed states and controls. This sequence is first used to learn the parameters of the non-linear function $\mathbf{f}$. The training data for the GP is then given by $\mathbf{x}_k = [\mathbf{s}(k), \mathbf{u}(k)]$ and $\mathbf{y}_k = \mathbf{s}(k+1) - \mathbf{s}(k) - \mathbf{f}(\mathbf{s}(k), \mathbf{u}(k))$.

V. Autonomous Blimp Control using Reinforcement Learning

Reinforcement learning based on policy search is frequently used in conjunction with a simulator to circumvent collecting large amounts of empirical data [10]. Simulator accuracy and speed as well as clever controller parametrization are the main factors in successful reinforcement learning. The controller must be flexible enough to learn complicated behavior, yet be simple enough to learn in a reasonable amount of time. In this section, we propose a parametrized yaw controller for the blimp. The objective of this controller is to steer the blimp from any yaw and yaw rate to a goal yaw, ψ^*, with zero yaw rate. While more complicated tasks could have been considered, this task was chosen to highlight the improvement offered by the GP-enhanced model.

Artificial Neural Networks (ANNs) are often chosen as a basis for controller design [11]. In this setting, the parameters of the controller are the weights of the neural network. If the desired controller behavior is well understood, however, then a more specific controller model can be easier to learn than a very general ANN model. With this in mind, the approach to controller parametrization taken here is based on linear time-optimal control theory and common sense. The optimal

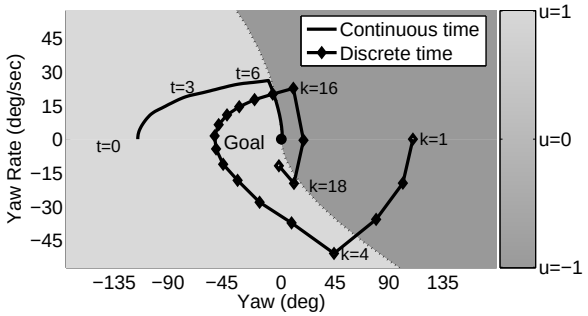

Fig. 3. Yaw rate vs. yaw is shown for continuous and discrete time simulations of the dynamics (1) using the continuous time optimal controller (with a zero order hold in the discrete time case). A time step of $dt = 0.8s$ was chosen to exaggerate the overshooting problem.

control problem is formulated for the second order linear approximation of the yaw dynamics,

$$\frac{d}{dt}\begin{bmatrix}\psi \\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}0 & 1 \\ 0 & -K_{drag}\end{bmatrix}\begin{bmatrix}\psi \\ \dot{\psi}\end{bmatrix} + \begin{bmatrix}0 \\ K_u^{pos}\end{bmatrix} u_t^{pos} + \begin{bmatrix}0 \\ K_u^{neg}\end{bmatrix} u_t^{neg}, \quad (24)$$

where ψ is the yaw error (the goal yaw, ψ^*, is taken to be zero without loss of generality) and $\dot{\psi}$ is the yaw rate. This model is a decent approximation for small pitch and roll angles. All gains are positive and two gains are needed for the tail motor because it is stronger in one direction than the other. If the tail command, u_t, is positive then $u_t^{pos} = u_t$ and $u_t^{neg} = 0$ (and vice versa). The tail motor command is limited to $u_t \in [-1, 1]$.

The cost functional in the optimal control formulation,

$$J(T, u) = \int_0^T \psi(\tau, u(\tau))^2 + w\dot{\psi}(\tau, u(\tau))^2 d\tau, \quad (25)$$

is minimal when the control drives yaw error and yaw rate to zero as fast as possible. The behavior can be adjusted by the scaling parameter w. The final time, T, is chosen to ensure the integrand can go to zero before T for realistic initial conditions. Linear optimal control theory suggests that with limited control authority, the tail motor should be run at full strength in a direction that depends on which side of a switching curve the state lies. An analytical expression for the switching curve can be found by solving the ODE (24) backwards in time from the origin with either full left or right control. The resulting controller is shown in the shading of Fig. 3.

This "optimal" controller works well on a continuous time system, however, it does a poor job of controlling yaw on the real blimp. One primary reason for this is that the blimp controller runs in $1/4sec$ discrete time. The blimp state tends to drift far across the switching curve before actually switching the control, resulting in overshooting. This can be seen in the comparison of continuous and discrete time simulations overlaid on Fig. 3. A second reason for poor performance is that the linear model does not account for non-linearity or noise.

The parametrized controller inspired by the continuous time optimal controller has a total of four parameters.

TABLE I

PREDICTION QUALITY (RMS ERROR)

Propagation method	$\mathbf{p}$(mm)	ξ(deg)	$\mathbf{v}$(mm/s)	ω(deg/s)	Time(s)
RK	7.6	0.55	25.1	2.18	0.025
ODE	7.6	0.55	25.3	2.11	0.289
RKGP	1.0	0.10	4.2	0.38	0.036
ODEGP	1.0	0.10	4.2	0.36	0.338
GPonly	1.3	0.15	5.3	0.81	0.012

The first three parameters are the gains in (24). The final parameter determines the smoothness of the controller near the switching curve. A smooth transition can alleviate the overshooting problem, see Fig. 3, caused by discrete time. Smoothing is achieved via a hyperbolic tangent function whose slope is determined by the fourth parameter.

Reinforcement learning based on policy search is used to find a locally stationary parameter set. Learning is initialized with hand selected parameters. Each set of parameters is evaluated by summing the integrand of the cost functional (25) along simulated trajectories. Note that multiple trajectories are often needed because the initial conditions are not fixed and because the simulation may include noise. When the model does include noise, the seed of the random number generator is fixed in accordance with PEGASUS [10].

VI. RESULTS

To test various aspects of our approach, the blimp is flown in the motion capture lab described in Section II. To learn predictive models of the blimp that are independent of the blimp's location and yaw in the lab, we removed all absolute coordinates from the blimp states and learn prediction models solely based on pitch, roll, and velocities represented in the blimp's coordinate frame.

Gaussian process calculations were done using Lawerence's FGPLVM package for Matlab [7]. The hyperparamters of the Gaussian process were optimized using a conjugate gradient optimization. We use the full Gaussian process with no approximations. Outputs from the Gaussian process are multivariate in that a single Gaussian process is used for calculating all output dimensions of the predictive model. All experiments use a discretization period of $1/4sec$.

A. Comparing Prediction Quality

This experiment is designed to test the prediction quality of the various models. The blimp is first flown to collect training data. This data is used to learn optimal parameters for the non-linear model. Additionally, the GP was used to learn residuals between the predictions of the non-linear model and the ground truth. A total of ~ 1000 training points are used here.

Prediction quality is assessed on additional test trajectories. From these trajectories we randomly choose states and predict the state one second into the future using different methods. The predicted state is then compared to the true state provided by the motion capture system. To perform prediction based on the non-linear model, we use Runge-Kutta (RK) and Matlab's `ode45` (ODE), which are different

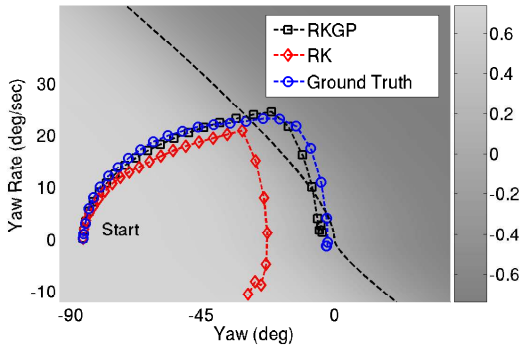

Fig. 4. RKGP trajectory prediction is much closer to ground truth trajectory.

numerical integration methods. RK is an extremely fast approximation, and `ode45` is less efficient but offers high accuracy using variable step sizes with tolerance checking. RKGP and ODEGP are mean predictions using the GP-enhanced approach to propagate the state of the blimp. Finally, a Gaussian process (GPonly) is used to predict the entire dynamics of the system - the non-linear model is not used at all. The training output learned by the GP in this case is the difference between successive ground truth states.

The results are summarized in Table I. The different columns present the root mean squared error of different components of the state, averaged over ~ 250 one second predictions. The first two rows show the accuracy using the numeric integration methods. The results using ODE are very similar to the results using RK only. The ODE predictor however takes on average 10 times longer than RK. We thus omit the ODE predictor from the other experiments. Results for prediction incorporating the GP (rows 3 and 4) show that learning residuals with the GP significantly improves prediction quality. The GPonly result shows that the zero mean Gaussian process does not fully extract all relevant aspects of the dynamical system from the training data. A likely source of error is also that the predictions of GPonly go to zero for samples drawn further from the training data.

Longer term prediction quality is illustrated in Fig. 4, which shows several sample turns of the blimp running the same series of motor commands. The trajectory predicted by RK deviates far from the real/ground truth trajectory whereas the RKGP trajectory matches the ground truth very well.

B. Trajectory Sampling with Noise

By incorporating noise into the GP predictions, trajectories can be sampled that have the same characteristics as real trajectories. To evaluate this capability, the blimp is run from extremely similar initial starting conditions (within human error) with the same motor commands for twenty trials. Using the starting condition and motor commands, the same trajectories are simulated either with RKGP using only the mean prediction, or with RKGP using both the mean prediction and its accompanying correlated noise (see Section IV-C). The distribution of the endpoints is then compared to the ground truth, as summarized in Table II. As expected, using the noise from the GP significantly increases the similarity between real and predicted data. Ignoring noise

TABLE II

ENDPOINTS OF TRAJECTORIES

Simulator	Mean Yaw (deg)	Mean Yaw Vel (deg/s)	σ Yaw (deg)	σ Yaw Vel (deg/s)
RKGP with noise	5.6	2.5	7.37	2.95
RKGP no noise	5.1	2.5	3.81	1.83
Ground Truth	5.8	.82	8.18	2.73

TABLE III

OPEN LOOP TURN ERROR ON REAL BLIMP

Simulator	Mean Yaw (deg)	Mean Yaw Vel (deg/s)	σ Yaw (deg)	σ Yaw Vel (deg/s)
RKGP	7.21	2.78	5.31	1.54
RK	31.1	13.2	5.58	1.44

results in underestimating the spread of the endpoints.

C. Reinforcement Learning for Blimp Control

Using the reinforcement learning techniques described in Section V, policies are learned for a 90° left turn, using both RK and RKGP for simulation. Each learned policy generates a sequence of motor commands that is transmitted to the blimp and executed. This is done in an open loop manner with no tracking information being sent back to the controller. The deviation from zero degree absolute yaw and zero degree/sec yaw velocity is recorded after the last command is sent. From Table III, the policy learned using RKGP is shown to be superior to the policy learned with RK alone. Several sample trajectories on the real blimp are shown in Fig. 5.

VII. CONCLUSION

We showed how to model the dynamics of an autonomous blimp using Gaussian process regression. GPs have several properties that make them ideally suited to this problem. They are non-parametric, which lets them model a wide range of dynamical systems. Furthermore, they can automatically learn the smoothness and noise levels of the underlying process, and they provide a notion of prediction uncertainty.

We performed various experiments controlling and predicting the motion of a real blimp in a motion capture lab. These experiments show that our GP model has significantly better predictive capabilities than standard non-linear models. The experiments also showed that sampling noisy trajectories from the GP model results in highly realistic simulations. Finally, we demonstrated that blimp control policies can be learned using reinforcement learning along with simulations provided by the GP model.

The performance of the GP prediction can be further improved by combining the GP with a non-linear model. In this combination, the GP learns to model those parts of the process dynamics that are not captured by the non-linear model. The resulting technique combines the benefits of both approaches: It incorporates prior knowledge about the dynamic process via the non-linear model and applies an extremely powerful regression model to learn improved predictions and noise models.

In future work, we will investigate GPs in the context of active experimentation, where the uncertainty of the GP is used to determine which type of control is needed to minimize the uncertainty in the predictive model. Furthermore, we will incorporate the GP model into a Kalman filter to track the blimp in a large-scale indoor environment.

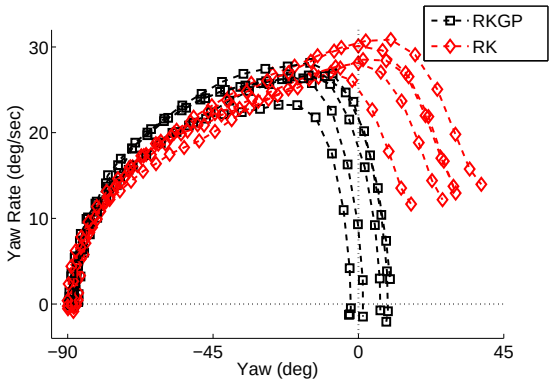

Fig. 5. Real runs with a policy learned using RKGP shows much higher accuracy than the policy with RK alone The goal is at (0,0).

ACKNOWLEDGMENT

This work was supported by the NSF under grant numbers IIS-0093406 and BE-0313250, and by DARPA's ASSIST and CALO Programs (contract numbers: NBCH-C-05-0137, SRI subcontract 27-000968).

REFERENCES

[1] Y. Bestaoui and S. Hima. Some insight in path planning of small autonomous blimps. *Archives of Control Sciences, Polish Academy of Sciences*, 11(3):21–49, 2001.

[2] T. Fukao, K. Fujitani, and T. Kanade. Image-based tracking control of a blimp. *Proc. of IEEE Conference on Decision and Control*, 2003.

[3] SBV Gomes and JG Ramos Jr. Airship dynamic modeling for autonomous operation. *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 4, 1998.

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

[5] E. Hygounenc, I.K. Jung, P. Souères, and S. Lacroix. The Autonomous Blimp Project of LAAS-CNRS: Achievements in Flight Control and Terrain Mapping. *The International Journal of Robotics Research*, 23(4-5):473–511, 2004.

[6] J. Kocijan and R. Murray-Smith. *Switching and Learning in Feedback Systems*, chapter Nonlinear Predictive Control with a Gaussian Process Model. Lecture Notes in Computer Science. Springer Verlag, 2005.

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

[8] RCGuys Radio Control Models. http://www.rcguys.com.

[9] R. M. Neal. *Baysesian Learning for Neural Networks*. Lecture Notes in Statistics 118, New York, 1996.

[10] A. Y. Ng and M. Jordan. PEGASUS:A policy search method for large MDPs and POMDPs. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 406–415, 2000.

[11] A.Y. Ng, H.J. Kim, M.I. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. *Advances in Neural Information Processing Systems (NIPS)*, 16, 2004.

[12] C.E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambirdge, Massachusetts, 2006.

[13] B.L. Stevens and F.L Lewis. *Aircraft Control and Simulation*. John Wiley & Sons, Inc, New York, NY, 1992.

[14] S. van der Zwaan, A. Bernardino, and J. Santos-Victor. Vision based station keeping and docking for an aerial blimp. *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[15] G.F. Wyeth and I. Barron. An Autonomous Blimp. *Proc. IEEE Int Conf. on Field and Service Robotics*, pages 464–470, 1997.

[16] H. Zhang and JP Ostrowski. Visual servoing with dynamics: control of an unmanned blimp. *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 1, 1999.