Gaussian Process Regression for Real-Time Estimate of the Thrust Curve for Quad Rotors with Interchangeable Blades

Wyatt McAllister[1]

***Abstract*— The paper lays out a Sparse Gaussian Process Regression architecture for the estimate of the thrust curves of rigid and flexible quadrotor blades whose dynamics are unknown. In this learning-framework, the dynamics of the quadrotor are separated from that for the rigid blade to expedite computation. This learning model is validated in simulation and benchmarked against a neural network and Non-Sparse Gaussian Process methods.**

I. Introduction

The large number of emerging consumer applications of quadrotors ([1], [2], [3], [4], [5]) are coupled with the growth in aftermarket modifications ([6], [7], [8], [9], [10]). Aftermarket parts are modifications that the flight controller was not optimized for, and this may result in a degradation of performance. Therefore, creating flight systems which can mitigate or reverse these affects is useful for the consumer market. Here we will focus on aftermarket modifications of the propeller, which may impact the performance of the UAV by changing the motor input to thrust output dynamics of the blade, or thrust curve. This paper explores a method for learning unknown blade dynamics via Sparse Gaussian Processes.

This research presents a regression architecture, summarized in Figure 1, which utilizes Sparse Gaussian Processes to estimate the thrust curve of an unknown blade in real time. This architecture separates the state to thrust dynamics from the dynamics for the blade. This allows for efficient computation by replacing the nonlinear dynamics of the quadrotor with a precomputed Gaussian Process. Our learning architecture then utilizes a thrust curve estimate which relies on operating data, where the thrust curve is unknown, and on the pre-trained thrust to state model, which may come from the same quadrotor flying with a blade with known dynamics. In this way, the real-time regression for the thrust curve can rely on an estimate which is trained on a larger dataset, leading to higher overall accuracy. The learning framework presented here is validated on a Crazyflie Simulator [11], which utilizes a control architecture derived in [12]. We first show that a perturbation from the thrust curve utilized in these works causes a significant deviation from the desired trajectory over that for known blade dynamics. We then benchmark our system against Neural Network and Non-Sparse Gaussian Process methods for the regression of rigid and flexible blades with unknown dynamics.

*This work was done towards completing the project component of ABE598

[1]Wyatt McAllister is with the Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign
wmcalli22@illinois.edu

II. Related Research and Contribution

This work will help allow quadrotors to utilize new blades with an unknown thrust curve by presenting an architecture for real-time learning of the blade dynamics. Previous work involving the estimation of quadrotor dynamics utilized a neural network approach, which integrated the dynamics of the blade [13]. Our approach separates the blade dynamics from the quadrotor dynamics. While we only retrain the model for the blade if the rotor is interchanged, the previous work would require the entire neural network to be retrained every time the blade was exchanged. Our architecture will help allow aftermarket modifications for the rotors to be utilized on quadrotors without compromising their performance. The Sparse Gaussian Process approach utilized here is computationally efficient, allowing this architecture to operate on devices with limited computational resources.

III. Overview

This paper is organized as follows. In section IV, we outline the control formulation of this problem. In section V, we present a theoretical explanation of Gaussian Processes, and present the motivation behind the utilization of the Sparse Gaussian Process framework for this application. In Section VI, we then present a detailed description of our algorithm within the Gaussian Process framework introduced. Finally, in Section VII, we present a detailed validation of our architecture, showing its performance for model regression as compared against Neural Network and Non-Sparse Gaussian Process approaches, and validating its impact on the performance of the quadrotor simulation utilized here. We also examine the performance of these methods for regressing the dynamics of flexible rotors, approximated as bounded and continuous stochastic perturbations from the blade rigid dynamics.

IV. Control Formulation

As mentioned previously, the quadrotor simulation used here, [11], is based on a control architecture detailed in [12]. In this control model, the motor command is computed based on the current and desired attitude, A_i, and A_d.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & a_1 + a_2\omega_r & a_1 \\ a_3 + a_4\omega_r & 0 & a_3 \\ a_5 & a_5 & 0 \end{bmatrix} \cdot \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} + \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & b_3 \end{bmatrix} \cdot \begin{bmatrix} U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{1}$$

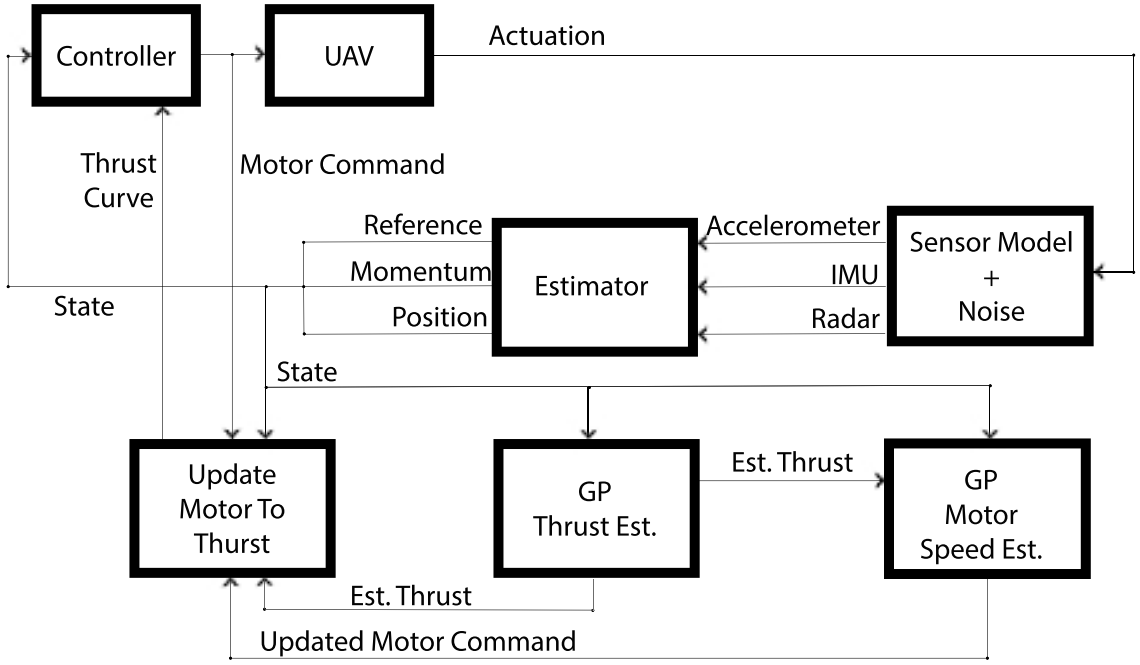


Fig. 1: This is the control architecture used for the UAV simulation in [11], where the controller actuates the systems through the UAV model. A state estimate is computed based on a sensor model. Our architecture then takes the state and computes the thrust estimate using a precomputed Gaussian process. An online Gaussian Process then computes the thrust curve, which is updated in the dynamics utilized by the control architecture.

In this model the constants $a_1, a_2, a_3, a_4, a_5, b_1, b_2, b_3$ are given in terms of the moments of inertia I_x, I_y, I_z and the inertia matrix in the body frame J_r.

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} \frac{I_y - I_z}{I_x} \\ \frac{-J_r}{I_x} \\ \frac{I_z - I_x}{I_y} \\ \frac{J_r}{I_y} \\ \frac{I_x - I_y}{I_z} \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{I_x} \\ \frac{1}{I_y} \\ \frac{1}{I_{zz}} \end{bmatrix} \tag{3}$$

The input commands U_2, U_3, U_4 are the control inputs detailed in [12] which are dependent on the dynamics for the blades. The estimate for the blade dynamics used in this controller is given by T_o. In this work, we show that our learning architecture improves the performance of the controller when the known blade dynamics are replaced with that of an unknown blade.

$$T_o(\omega) = \omega^2 \tag{4}$$

V. Background on Gaussian Processes

This section presents the theoretical background for the Gaussian Process methods utilized in this paper. In section V-A, we present a theoretical explanation of Gaussian Process regression. Then, in Section VII-B, we discuss the methodology behind the Sparse Gaussian Process framework, and explain how this expedites computation over Gaussian Process methods which utilize the full set of basis.

A. GP Regression

Let s be the set of state measurements.

$$S_\tau = \{s_1, ..., s_\tau\}$$

In this case the state is given by the pitch, roll, and yaw, ϕ, θ, φ, and the linear and angular acceleration of the quadrotor within the earth frame, $\ddot{x}, \ddot{y}, \ddot{z}, \ddot{\phi}, \ddot{\theta}, \ddot{\varphi}$.

$$s = \begin{bmatrix} \phi & \theta & \varphi & \ddot{\phi} & \ddot{\theta} & \ddot{\varphi} & \ddot{x} & \ddot{y} & \ddot{z} \end{bmatrix} \tag{5}$$

We then write the thrust, $T(s)$, as a Gaussian process.

$$T(s) \sim GP\left(m(s), k(s, s')\right) \tag{6}$$

For each state measurement, the observed output is given by the mean for each sample plus a Gaussian noise term ϵ.

$$y_{z_i} = m(s_i) + \varepsilon_i, \varepsilon_i \sim N\left(0, \omega^2\right) \tag{7}$$

Therefore, the state measurements are transformed to observed system outputs, and our desire is to extract the thrust estimate from tis data.

$$s_\tau = \{s_1, ..., s_\tau\} \rightarrow y = \{y_1, ..., y_\tau\}^T$$

The set of possible states is an infinite dimensional function space, in this case a Hilbert Space. Within the Gaussian Process framework, this space is spanned be a set of kernel functions, which each have Gaussian distribution.

$$k(s, s_o) = e^{\left(-\frac{\|s - s_o\|}{2\sigma^2}\right)} \tag{8}$$

We define the following quantities in terms of the kernel functions: K, the covariance matrix up to time τ, $k_{s_{\tau+1}}$, the joint covariance of the new measurement with that of the previous sequence, and $k^*_{s_{\tau+1}}$, the covariance of the new measurement.

$$K_{ij} = k\left(s_i, s_j\right) \tag{9}$$

$$k_{s_{\tau+1}} = K\left(s_\tau, s_\tau\right) \tag{10}$$

$$k^*_{s_{\tau+1}} = k\left(s_{\tau+1}, s_{\tau+1}\right) \tag{11}$$

Each new data point, $s_{\tau+1}$, will then lead to a resultant update in the posterior Gaussian distribution for the data. Each new observation $y_{\tau+1}$ is jointly distributed with the previous observation sequence y^τ.

$$\begin{bmatrix} y_\tau \\ y_{\tau+1} \end{bmatrix} \sim N\left(0, \begin{bmatrix} K\left(s_\tau, s_\tau\right) + \omega^2 I & k_{s_{\tau+1}} \\ k^T_{s_{\tau+1}} & k^*_{s_{\tau+1}} \end{bmatrix}\right) \tag{12}$$

It is known that the conditional distribution of the latest observation given the previous observation sequence is also Gaussian.

$$P\left(y_{\tau+1} | \, s_\tau, y_\tau, s_{\tau+1}\right) \sim N\left(\hat{m}_{\tau+1}, \hat{\Sigma}_{\tau+1}\right) \tag{13}$$

This distribution is written in terms of the best estimate $\hat{m}_{\tau+1}$ and the error covariance $\hat{\Sigma}_{\tau+1}$.

$$\hat{m}_{\tau+1} = \beta^T_{\tau+1} k_{s_{\tau+1}} \tag{14}$$

$$\hat{\Sigma}_{\tau+1} = k^*_{s_{\tau+1}} - k^T_{s_{\tau+1}} C_\tau k_{s_{\tau+1}} \tag{15}$$

These in tern involve the inverse covariance of the observation sequence, C_τ, and the product of this term with the new observation, $\beta_{\tau+1}$.

$$C_\tau = \left(K\left(s_\tau, s_\tau\right) + \omega^2 I\right)^{-1} \tag{16}$$

$$\beta_{\tau+1} = C_\tau y_\tau \tag{17}$$

B. Motivation for Sparse Gaussian Processes

As one can observe from the constants above, the inverse must be computed in this scenario. Computing the Gaussian Process with the full basis set requires a computation time of $O(N^3)$ [14], where N is the total size of the training data. We therefore desire a method to expedite computation by pruning the space of basis while maintaining an effective state estimate given all the current data. Gaussian Process Algorithms which realize this are called Sparse Gaussian Processes. Furthermore, since we desire a model that trains in real-time, we need a Sparse Gaussian Process Algorithm which is optimized for an online setting, whiles rules out offline Sparse Gaussian Process approaches such as ([15], [16]). An Online Sparse GP Algorithm is Csato's Algorithm [14], which is shown to reduce the computation time from $O(N^3)$ to $O(Np^2)$, where N is the total size of the training data and p is the size of the basis set. Furthermore, the online nature of this algorithm requires that it only make one pass through the dataset to train.

We now provide a mathematical explanation of why the Sparse Gaussian Process framework provides added efficiency without compromising estimation accuracy. We define the coefficient α_τ to be the stochastic projection of the new data onto the old within the kernel space, as given by the known best estimate for the Gaussian process.

$$\alpha_\tau = K^{-1}_{S_\tau} k_{s_{\tau+1}} \tag{18}$$

We define the length of the residual $\gamma_{\tau+1}$, to be the distance between the basis of the new data point, and the sum of the projections onto the other bases functions as given by the product of the coefficients α with each basis.

$$\gamma_{\tau+1} = \min_{\alpha_i} \left\| \sum_{i=1}^{\tau} \alpha_i \psi(z_i) - \psi\left(s_{\tau+1}\right) \right\|^2 \tag{19}$$

Matching the components of this expression with the known parameters in the Gaussian process, we arrive at an expression for $\gamma_{\tau+1}$ which includes the known parameters.

$$\gamma_{\tau+1} = k^*_{\tau+1} - k^T_{s_{\tau+1}} \alpha_\tau \tag{20}$$

Our desire is to utilize an algorithm which chooses a basis set that results in the largest overall stochastic difference between the basis vectors. Sparse Gaussian Process achieve this goal. These algorithms first add the candidate basis to the basis set. Then, they test the KL divergence of all subsets of this space where one of the basis is removed, and discard the basis that results in the minimum KL divergence when removed.

Our implementation, detailed in the following section, presents a two-tiered architecture which utilizes Csato's Algorithm to compute a Gaussian Process estimate of the thrust curve based on another pre-computed Gaussian Process for the state to thrust relationship.

VI. Algorithm for Thrust Curve Estimate

As discussed in the previous section, we compute an estimate for the Thrust based on the current state data utilizing a Gaussian Process Regression Model. This estimate is defined to be the mean of the Gaussian Process.

$$T(s) \sim GP\left(\hat{m}_T(s), k_T(s,s)\right) \Rightarrow \hat{T}(s) = \hat{m}_T(s) \quad (21)$$

This model is trained with an existing UAV Control simulation for the Crazyflie Quadrotor [17]. For these experiments, several different thrust curves were generated by perturbing the existing quadratic thrust curve, $T(\omega)$, with linear and constant terms, which may represent a different thickness, length, or weight of the blade ([17], [18]). This function represents an unknown blade dynamics after the blade has been changed. Our assumption is that the dynamics relating the state to the thrust, $\hat{m}_T(s)$, can be trained independently of the relationship between the motor input and the thrust, $\hat{m}_\omega$. Based on the regression framework described in the previous section, the state to thrust dynamics, $\hat{m}_T(s)$, is known. We therefore use a nested Gaussian Process architecture for the estimation of the relationship between the motor speed and the thrust, $\hat{m}_\omega$. We start with the state data, s, for the system with the unknown blade.

$$s = \begin{bmatrix} \phi & \theta & \varphi & \ddot{\phi} & \ddot{\theta} & \ddot{\varphi} & \ddot{x} & \ddot{y} & \ddot{z} \end{bmatrix} \quad (22)$$

Our new Gaussian process regression model for the motor command will include a composite state, s, which is composed of the thrust estimate for the updated state data along with the data itself.

$$s' = \begin{bmatrix} \hat{m}_T(s) & s \end{bmatrix} \quad (23)$$

To estimate the motor command, ω, we now use a Gaussian Process Regression Model which is identical to that for the thrust. However, the state here is replaced by the composite state, s'.

$$\omega\left(s'\right) \sim GP\left(\hat{m}_\omega\left(s'\right), k\omega\left(s', s_o\right)\right) \Rightarrow \hat{\omega}(s) = \hat{m}_\omega\left(s'\right) \quad (24)$$

To implement this architecture, summarized in Figure 1, we utilize the regression framework described in this section and summarized in Algorithm 1 Below. The software for this algorithm was adapted from the implementation of Csato's algorithm provided by [19].

Algorithm 1 Algorithm 1

1: INPUT: Gaussian Process Regression Model for Thrust $T(s) = \hat{m}_T(s)$, updated state data s.
2: Compute the thrust estimate for the state data $\hat{m}_T(s)$.
3: Define the composite state to be:

$$s' = \begin{bmatrix} \hat{m}_T(s) & s \end{bmatrix} \quad (25)$$

4: Compute the Gaussian Process Regression for the Motor Command, ω.

$$\omega\left(s'\right) \sim GP\left(\hat{m}_\omega\left(s'\right), k\omega\left(s', s_o\right)\right) \Rightarrow \hat{\omega}(s) = \hat{m}_\omega\left(s'\right) \quad (26)$$

VII. Results

In this section, we test this model on the simulated control architecture utilized here. First, in section VII-A, we show how our parameter estimation framework improves the performance of the Crazyflie quadrotor simulation [11] used here for unknown blade dynamics. Second, in section VII-B, we test our learning architecture on several different sets of blade dynamics. Then, in section VII-C, we show that the accuracy of the regression for the thrust dynamics is not affected by the blade dynamics the system was trained on. In Section VII-D, we compare our systems performance to a system which utilizes a single hidden layer neural network. In Section VII-E, we repeat this comparison for a regression using Non-Sparse Gaussian Process from Rasmussen [20]. In section VII-F, we examine the performance of these three architectures for the regression of the dynamics of a flexible blade. Finally, in VII-G, we show that the results of the previous experiment are not changed significantly when the thrust curve estimator utilizes an estimate for the thrust dynamics trained on the unknown blade instead of the known blade.

A. Validation of Control Architecture

In this section, we replace the known dynamics of the blade, T_o (repeated below), with those of an unknown blade. We then confirm that the tracking error for the attitude control is improved when an online GP estimate for the new blade dynamics is utilized instead of the previous estimate for the blade dynamics.

$$T_o(\omega) = \omega^2 \quad (27)$$

The desired trajectory utilized for these experiments is shown in Figure 3, where the control has knowledge of the correct thrust curve given by T_o. In our experiment, we first compute average deviation from the desired trajectory or tracking error, e, when the correct thrust curve is known. We then replace the blade dynamics in the simulation with those of an unknown blade. We re-compute the deviation again when the controller utilizes our Gaussian process estimate for the actuated thrust curve, T_{new}. Next, we compute the deviation when T_{new} is actuated but the invalid estimate of T_o is utilized in control. These results are summarized in Table 1 below.

$$T_{new}(\omega) = \omega^2 + 500 \cdot \omega + 500 \quad (28)$$

$$e = \frac{\sqrt{\left(x - x_{ref}\right)^2 + \left(y - y_{ref}\right)^2 + \left(z - z_{ref}\right)^2}}{\sqrt{x_{ref}^2 + y_{ref}^2 + z_{ref}^2}} \quad (29)$$

Table 1: Performance of Controller

Simulation	$T_{true}(\omega)$	$T_{est.}(\omega)$	e
Test 1	T_o	T_o	0.30050
Test 2	T_{new}	$\hat{\mathrm{T}}_{\mathrm{new,GP}}$	0.31810
Test 3	T_{new}	T_o	0.30078

As shown in equations 29 and 30, our estimate for the thrust curve improves the tracking error for the unknown dynamics from approximately four percent of what it was for known dynamics to under one percent. This performance improvement shows that our learning positively model impacts the attitude control for this experiment.

$$\Delta e_o = \frac{e_{known} - e_{unknown}}{e_{known}} = \frac{|0.3050 - 0.3181|}{0.3050} = 4.29\% \tag{30}$$

$$\Delta e' = \frac{e_{known} - e_{est}}{e_{known}} = \frac{|0.3050 - 0.3078|}{0.3050} = 0.918\% \tag{31}$$

The position in this quadrotor simulation is measured in meters. Therefore, as shown in equation 32, the tracking error is improved by approximately 2 cm.

$$\Delta e = 31.81 - 30.078 = 1.732 \tag{32}$$

This improvement is significant for applications where several quadrotors may operate close to humans [4].

B. Validation of Learning Architecture for Thrust Curve

We assume model accuracy for the Gaussian process regression to be within a certain process noise ϵ. For the thrust model, we write the thrust prediction T^p, in terms of the desired thrust from the simulation T^d, and the process noise.

$$T^p = T^d + \varepsilon_T \Rightarrow \varepsilon_T = \left| T^p - T^d \right| \tag{33}$$

The process noise is approximated by the root mean square error for the model.

$$\hat{\varepsilon}_T \approx \sqrt{\left(\hat{T}^p - \hat{T}^d\right)^2} = RMSE_T \tag{34}$$

The estimator, and process noise for the motor command are written similarly.

$$\omega^p = \omega^d + \varepsilon_\omega \Rightarrow \varepsilon_\omega = \left| \omega^p - \omega^d \right| \tag{35}$$

$$\hat{\varepsilon}_\omega \approx \sqrt{\left(\hat{\omega}^p - \hat{\omega}^d\right)^2} = RMSE_\omega \tag{36}$$

We now examine the four thrust curves summarized in Table 1. The process noise for each regression is shown. Furthermore, the error at each state, as defined in equations 35 and 36, is plotted alongside the regressed functions. These plots are shown in Figures 3, 4, 5, and 6 for Blades 1, 2, 3, and 4 respectively.

$$E = \frac{T^p - T^d}{T^d} \tag{37}$$

$$E_\omega = \frac{\omega^p - \omega^d}{\omega^d} \tag{38}$$

Table 1: Performance with SGP for Dynamic Inversion

Simulation	$T(\omega)$	$RMSE_T$	$RMSE_\omega$
Blade 1	$\omega^2 + \omega + 1$	0.0109	0.0056
Blade 2	$\omega^2 + 5 \cdot \omega + 5$	0.0106	0.0057
Blade 3	$\omega^2 + 50 \cdot \omega + 50$	0.0108	0.0057
Blade 4	$\omega^2 + 500 \cdot \omega + 500$	0.0111	0.0057

The estimate of the process noise ϵ, is the root mean square error above. Therefore, these results show that we expect the estimate of the thrust, T^p, to be within approximately 0.01 N from the desired thrust, T^d, and the estimate for the motor command, ω^p, to be within a deviation of just over 0.005 RPM from the desired motor command, ω^d. The following sections will demonstrate that the Gaussian Process estimator for the state to thrust relationship may be trained on a different dataset from which it is tested, and show the utility of this estimation framework for UAV control.

C. Examination of Performance for Pre-Trained Dynamic Model

The results for Blades 1, 2, 3, and 4 as shown in Table 1 and Figures 4, 5, 6, and 7, show how the regression performs on the given data sets. We now examine how this performance changes when the thrust model is pre-trained on a different data from the one it is tested on.

For this experiment, we pre-train the Gaussian Process estimator for the state the thrust relationship on the data given by the thrust curve, $T_{train}(\omega)$, and then validate it on the data set given by the thrust curve for Blades 1, 2, 3 and 4, $T_{blade}(\omega)$. As summarized in Table 2, the regression model is tested on the new state data with the updated thrust curves for Blades 1, 2, 3 and 4. The results are summarized in Table 2, where it is observed that the accuracy is almost identical to that trained with the previous thrust estimate. This confirms that the decoupling of the thrust to state dynamics from the thrust to motor command dynamics does not affect the overall regression for the motor command. This confirms that, in the system considered, regression will be robust to the replacement of rotor blades when using a pre-trained model for the state to thrust dynamics.

$$T_{train}(\omega) = \omega^2 \tag{39}$$

$$T_{Blade,1}(\omega) = \omega^2 + \omega + 1 \tag{40}$$

$$T_{Blade,2}(\omega) = \omega^2 + 5 \cdot \omega + 5 \tag{41}$$

$$T_{Blade,3}(\omega) = \omega^2 + 50 \cdot \omega + 50 \tag{42}$$

$$T_{Blade,4}(\omega) = \omega^2 + 500 \cdot \omega + 500 \tag{43}$$

Table 2: Performance of Pre-Trained Dynamic Model

Simulation	$RMSE_{T_{train}}$	$RMSE_{T_{test}}$
Blade 1	0.0108	0.0109
Blade 2	0.0107	0.0106
Blade 3	0.0108	0.0108
Blade 4	0.0111	0.0111

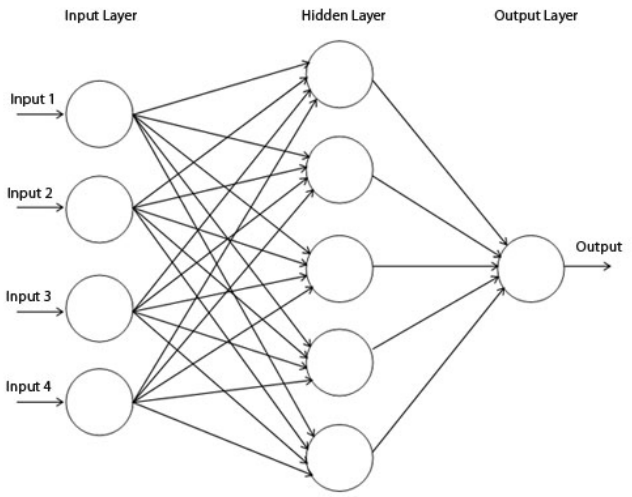


Fig. 2

D. Comparison Against Neural Network

In this section, an inversion of the dynamics for the four blades, as shown in Table 1, is done with a Single Hidden Layer Neural Network. A sketch of a typical single hidden layer neural network is shown in [22], and repeated in Figure 2. Unlike Figure 2, we have ten hidden neurons and ten inputs corresponding to the states given by equations 18 and 19.

This network utilizes ten hidden neurons, a sigmoidal activation function given in equation 42, an output equation given by equation 43, and weight update equations given by equations 44 and 45.

$$v = \frac{1}{1 + e^{-\sum_{i=1}^{n} X_i W_i}} \tag{44}$$

$$y^p = y^p + v \cdot W_y \tag{45}$$

$$W'_i = W_i + \eta \cdot v \cdot (1 - v) \cdot \left(y^d - y^p\right) \cdot W_y \cdot X_i \tag{46}$$

$$W'_y = W_y + \eta \cdot v \cdot \left(y^d - y^p\right) \tag{47}$$

This network trains over two hundred epochs. The root mean square error for each epoch is computed. Process noise for the experiments with each blade is given by the minimum of the root mean square error over all the epochs, as summarized in Table 3 below, where the left column shows the process noise for the neural network approach and the right side (repeated from Table 1) shows a comparison with the nested Sparse Gaussian Process approach. Figures 8, 9, 10 and 11 plot the predicted motor command, ω^p, against the desired command, ω^p, and show this plot above that charting the RMSE for each epoch.

$$\hat{\varepsilon}_\omega = \min_{epochs} RMSE_{\omega,epoch} \tag{48}$$

Table 3: The RMSE for the neural network approach is in the column for $RMSE_{\omega_{NN}}$. This is compared to the RMSE for the Sparse Gaussian Process approach $RMSE_{\omega_{SGP}}$. Note that $RMSE_{\omega_{SGP}}$ is repeated from Table 1.

Simulation	$RMSE_{\omega_{NN}}$	$RMSE_{\omega_{SGP}}$
Blade 1	0.0032	0.0056
Blade 2	0.0031	0.0057
Blade 3	0.0031	0.0057
Blade 4	0.0048	0.0057

As can be seen in Table 3, the neural network approach outperforms the error of the Sparse GP by approximately fifty percent.

We, then benchmark the overall system when our Sparse Gaussian Process estimator for the motor command, ω, is computed based on an estimate for the thrust provided by the above neural network. From here forward, we will refer to this as an NN-SGP architecture.

Table 4: The RMSE for the NN-SGP, is in the column for $RMSE_{\omega_{NN}}$. This is compared to the RMSE for the nested Sparse Gaussian Process approach $RMSE_{\omega_{SGP}}$. Note that $RMSE_{\omega_{SGP}}$ is repeated from Table 1.

Simulation	$RMSE_{\omega_{NN}}$	$RMSE_{\omega_{SGP}}$
Blade 1	0.0060	0.0056
Blade 2	0.0058	0.0057
Blade 3	0.0072	0.0057
Blade 4	0.0059	0.0057

Table 4 shows that the nested Gaussian Process architecture delivers superior performance to the hybridized neural network approach on this dataset. This may be because of the limited available data or because of the depth of the network.

Finally, we benchmark the performance of our overall system against a nested neural network structure, where the first neural network computes the state to thrust dynamics, and these are utilized by the second neural network to compute the blade dynamics.

Table 5: The RMSE for nested neural network is in the column for $RMSE_{\omega_{NN}}$. This is compared to the RMSE for the nested Sparse Gaussian Process approach $RMSE_{\omega_{SGP}}$. Note that $RMSE_{\omega_{SGP}}$ is repeated from Table 1.

Simulation	$RMSE_{\omega_{NN}}$	$RMSE_{\omega_{SGP}}$
Blade 1	0.0060	0.0056
Blade 2	0.0059	0.0057
Blade 3	0.0060	0.0057
Blade 4	0.0089	0.0057

Table 5 shows that the nested Sparse Gaussian Process architecture delivers superior performance to the nested neural network approach on this dataset. This may be because of the limited available data of compounding in the error between the two networks.

E. Comparison Against Non-Sparse Gaussian Process

In this section, an inversion for the dynamics of the four blades, shown in Table 1, is done with a Non-Sparse Gaussian Process. This Gaussian Process is a direct implementation of the Gaussian Process regression detailed in Section V-A, utilizing a regression implementation detailed in [20]. The RMSE for each inversion is shown in Table 6. The Gaussian Process output is plotted in Figures 11, 12, 13, and 14.

Table 6: The RMSE for GP is in the column for $RMSE_{\omega_{GP}}$. This is compared to the RMSE for the nested Sparse GP approach $RMSE_{\omega_{SGP}}$. Note that $RMSE_{\omega_{SGP}}$ is repeated from Table 1.

Simulation	$RMSE_{\omega_{GP}}$	$RMSE_{\omega_{SGP}}$
Blade 1	1.3303e-04	0.0056
Blade 2	1.3222e-04	0.0057
Blade 3	1.2776e-04	0.0057
Blade 4	1.2992e-04	0.0057

Table 6 shows that the Non-Sparse Gaussian Process delivers superior performance for inversion of the blade dynamics, as is to be expected.

We again benchmark the overall system when our Sparse Gaussian Process estimator for the motor command, ω, is computed based on an estimate for the thrust dynamics computed with the above Non-Sparse GP implementation. From here forward, we refer to this as a GP-SGP architecture. These results are shown in Table 7.

Table 7: The RMSE for the GP-SGP is in the column for $RMSE_{\omega_{GP}}$. This is compared to the RMSE for the nested Sparse Gaussian Process approach $RMSE_{\omega_{SGP}}$. Note that $RMSE_{\omega_{SGP}}$ is repeated from Table 1.

Simulation	$RMSE_{\omega_{GP}}$	$RMSE_{\omega_{SGP}}$
Blade 1	0.0059	0.0056
Blade 2	0.0058	0.0057
Blade 3	0.0059	0.0057
Blade 4	0.0058	0.0057

Table 7 shows that the hybrid approach with the Non-Sparse Gaussian Process does not improve performance. As shown in Table 8, this is not a result of overfitting in the Non-Sparse GP.

Finally, we benchmark the performance of our overall system against a nested Non-Sparse Gaussian Process network structure, where the first GP computes the state to thrust dynamics, and these are utilized by the second neural network to compute the blade dynamics.

Table 8: The RMSE for nested GP is in the column for $RMSE_{\omega_{GP}}$. This is compared to the RMSE for the nested Sparse Gaussian Process approach $RMSE_{\omega_{SGP}}$. Note that $RMSE_{\omega_{SGP}}$ is repeated from Table 1.

Simulation	$RMSE_{\omega_{GP}}$	$RMSE_{\omega_{SGP}}$
Blade 1	1.2854e-04	0.0056
Blade 2	1.3123e-04	0.0057
Blade 3	1.3059e-04	0.0057
Blade 4	1.2857e-04	0.0057

Table 8 shows that the nested Gaussian Process architecture delivers superior performance to the nested Sparse Gaussian Process approach on this dataset.

F. Performance of Algorithms for Systems with Deformable Blades

Allowing quadrotors to adapt to deformable blades with unknown dynamics is useful for applications where the quadrotor will operate near humans [4]. Past work has shown that the dynamic response of deformable blades is bounded [21]. To model a flexible rotor, we desire noise which is both bounded and continuous. For continuity, we choose, a Brownian motion, δ, as our noise model. For boundedness, we model the perturbation of the flexible blade as a cosine of the Brownian motion. The resulting thrust, T_{deform}, is shown in equation 43.

$$T_{deform}(\omega) = \omega^2 + A \cdot \cos(\delta), \delta \sim N(0, t) \tag{49}$$

In this experiment, we first train the thrust dynamics for our learning system on the rigid blade, T_{rigid}. Then, we benchmark the performance of the regression for the thrust curve of T_{deform}, on the nested NN, GP, and SGP approaches with varying amplitude. These results are shown in Table 9.

$$T_{rigid}(\omega) = \omega^2 \tag{50}$$

Table 9: The RMSE for nested GP is in the column for $RMSE_{\omega_{GP}}$. The RMSE for nested NN is in the column for $RMSE_{\omega_{NN}}$. The RMSE for the nested Sparse Gaussian Process is in the column for $RMSE_{\omega_{SGP}}$.

A	$RMSE_{\omega_{GP}}$	$RMSE_{\omega_{NN}}$	$RMSE_{\omega_{SGP}}$
1	1.3185e-04	0.0031	0.0059
5	1.3225e-04	0.0032	0.0060
50	1.3051e-04	0.0031	0.0058
500	1.2289e-04	0.0031	0.0059

Table 9 shows that these learning architectures have comparable performance to when they were tested on the models for the rigid blades summarized in Tables 1, 3, and 6.

G. Validation of Pre-Trained Thrust Model for Learning Architectures

Here, unlike the previous section, the regressions are trained on the deformable blade dynamics. These results are shown in Table 10. This will allow us to benchmark the performance of our learning architectures against a system

using a thrust model which is pre-trained on the rigid blade dynamics, as in the previous section.

Table 10: The RMSE for nested GP is in the column for $RMSE_{\omega_{GP}}$. The RMSE for nested NN is in the column for $RMSE_{\omega_{NN}}$. The RMSE for the nested Sparse Gaussian Process is in the column for $RMSE_{\omega_{SGP}}$.

A	$RMSE_{\omega_{GP}}$	$RMSE_{\omega_{NN}}$	$RMSE_{\omega_{SGP}}$
1	1.2816e-04	0.0032	0.0059
5	1.2893e-04	0.0032	0.0060
50	1.3225e-04	0.0031	0.0058
500	1.2287e-04	0.0031	0.0059

To compare the performance of the learning architectures when they utilize different blade dynamics to train the thrust model, we calculate the percent change in the RMSE, Δ, between Tables 9 and 10, shown in Table 11 below.

Table 11: The percent change in the RMSE for nested GP is in the column for $\Delta_{\omega_{GP}}$. The Percent change in the RMSE for nested NN is in the column for $\Delta_{\omega_{NN}}$. The percent change in the RMSE for the nested Sparse Gaussian Process is in the column for $\Delta_{\omega_{SGP}}$.

A	$\Delta_{\omega_{GP}}$	$\Delta_{\omega_{NN}}$	$\Delta_{\omega_{SGP}}$
1	-2.879	+3.125	0
5	-2.575	0	0
50	+1.315	0	0
500	-0.0162	0	0

Table 11 shows that the percent change in the RMSE for these architectures is small when the pre-trained thrust model replaces the thrust model trained with the data resulting from the new blade with unknown dynamics. The relatively small percentage changes, all below 4%, given the high amplitude of the stochastic term, validates the separation of the blade dynamics and quadrotor dynamics for these architectures. We note that the Non-Sparse GP decreases for one of the datasets, whereas the error for the Sparse GP does not measurably change. Even though the Sparse GP has a much larger error than the Non-Sparse GP, its results are more consistent across multiple trails.

VIII. Conclusion

These results show that, in this simulated set of experiments, our learning model is accurate for the regression of the dynamics of unknown blades. This accomplishes the goal of this project, which was to compare the performance of the Sparse Gaussian process regression for the regression of the thrust to motor input relationship of unknown blades with that of neural networks and Non-Sparse Gaussian Processes. It was shown that this architecture augments an existing UAV control previously simulated on the Crazyflie by estimating an unknown thrust curve. This learning model utilized a nested Sparse Gaussian Process framework, which took advantage of the dynamics of the UAV to allow computation of the thrust curve of the unknown blade using a thrust to state model pre-trained with a known blade.

References

[1] "FreeSkies Takes Rein of the Quadcopter", Engineering.illinois.edu, 2017. [Online]. Available: http://engineering.illinois.edu/news/article/10871. [Accessed: 05- May- 2017].

[2] "Drone Janus 360 - Dronevolt", Dronevolt, 2017. [Online]. Available: http://www.dronevolt.com/en/expert-solutions/drone-janus-360/. [Accessed: 07- May- 2017].

[3] E. Mangina "Drones for live streaming of visuals for people with limited mobility," ICVSM

[4] "NSF Award Search: Award1525900 - NRI: Collaborative Research: ASPIRE: Automation Supporting Prolonged Independent Residence for the Elderly", Nsf.gov, 2017.

[5] "How drones help us study our climate, forecast weather" [Online]. Available: http://cen.acs.org/articles/94/i9/drones-help-us-study-climate.html

[6] DJI-Carbon Fiber Self Tightening Propellers, Store.dji.com, 2017. [Online]. Available: https://store.dji.com/product/9450-carbon-fiber-self-tightening-rotor-black. [Accessed: 07- May- 2017].

[7] "Drone World's Range Extender Antenna System: How to Maximize the Phant", Drone-world.com, 2017.[Online].Available:http://www.drone-world.com/drone-world-maximized-long-range-phantom-3-system.

[8] "Lume Cube Lighting Kit for DJI Quadcopter Drones : Cabela's", www.cabelas.com, 2017. [Online]. Available: http://www.cabelas.com/product/hobbies/drones-and-drone-accessories—/pc/104799780/c/671030280/null/2392542.uts. [Accessed: 07- May- 2017].

[9] S. Kit, "Buy Sentera Phantom 3 NDVI Camera Upgrade Kit - Rise Above Australia", Rise Above Custom Drones, 2017. [Online]. Available: http://www.riseabove.com.au/sentera-phantom-3-ndvi-camera-upgrade-kit. [Accessed: 07- May- 2017].

[10] Panel EN - Edito - Parrot S.L.A.M.dunk", Parrot Store Official, 2017. [Online]. Available: https://www.parrot.com/us/business-solutions/parrot-slamdunkparrot-slamdunk. [Accessed: 07- May- 2017].

[11] A. Gibiansky, "Quadcopter Dynamics and Simulation - Andrew Gibiansky", Andrew.gibiansky.com, 2017. [Online]. Available: http://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/.

[12] C. Dikmen, A. Arisoy and H. Temeltas, "Attitude control of a quadrotor," 2009 4th International Conference on Recent Advances in Space Technologies, Istanbul, 2009, pp. 722-727.

[13] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," 2016 IEEE 55th Conference on Decision and Control (CDC), Las Vegas, NV, 2016, pp. 4653-4660.

[14] Csat and M. Opper, "Sparse On-Line Gaussian Processes," in Neural Computation, vol. 14, no. 3, pp. 641-668, March 1 2002.

[15] J. Candela and C. Rasmussen, A unifying view of sparse approximate Gaussian process regression, J. Mach. Learn. Res., vol. 6, pp. 19391959, Dec. 2005.

[16] M. Gredilla, J. Candela, C. Rasmussen, and A. Figueiras-Vidal, Sparse spectrum Gaussian process regression, J. Mach. Learn. Res., vol. 11, pp. 18651881, Jun. 2010.

[17] L. Yi, "The dynamical model of rotor blade system," 2016 18th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, 2016, pp. 335-338.

[18] Y. Wang, H. Wang and Z. Gao, "Dynamic modeling of helicopter rotor blades," in Tsinghua Science and Technology, vol. 14, no. S2, pp. 84-88, Dec. 2009.doi: 10.1016/S1007-0214(10)70037-7

[19] [Online]. Available: http://web.mit.edu/girishc/www/resources/ resources20files. 2017.

[20] Rasmussen, C. E. and C. K. I. Williams. Gaussian Processes for Machine Learning. MIT Press. Cambridge, Massachusetts, 2006.

[21] S. Bosnyakov, A. Bykov, V. Coulech, S. Fonov, A. Morozov and V. Moskalik, "Blade deformation and PSP measurements on the large-scale rotor by videometric system," Instrumentation in Aerospace Simulation Facilities, 1997. ICIASF '97 Record., International Congress on, Pacific Grove, CA, 1997, pp. 95-104.

[22] "Neural Networks OpenCV 2.4.13.2 documentation", Docs.opencv.org, 2017. [Online]. Available: http://docs.opencv.org/2.4/modules/ml/doc/neuralnetworks.html.

.

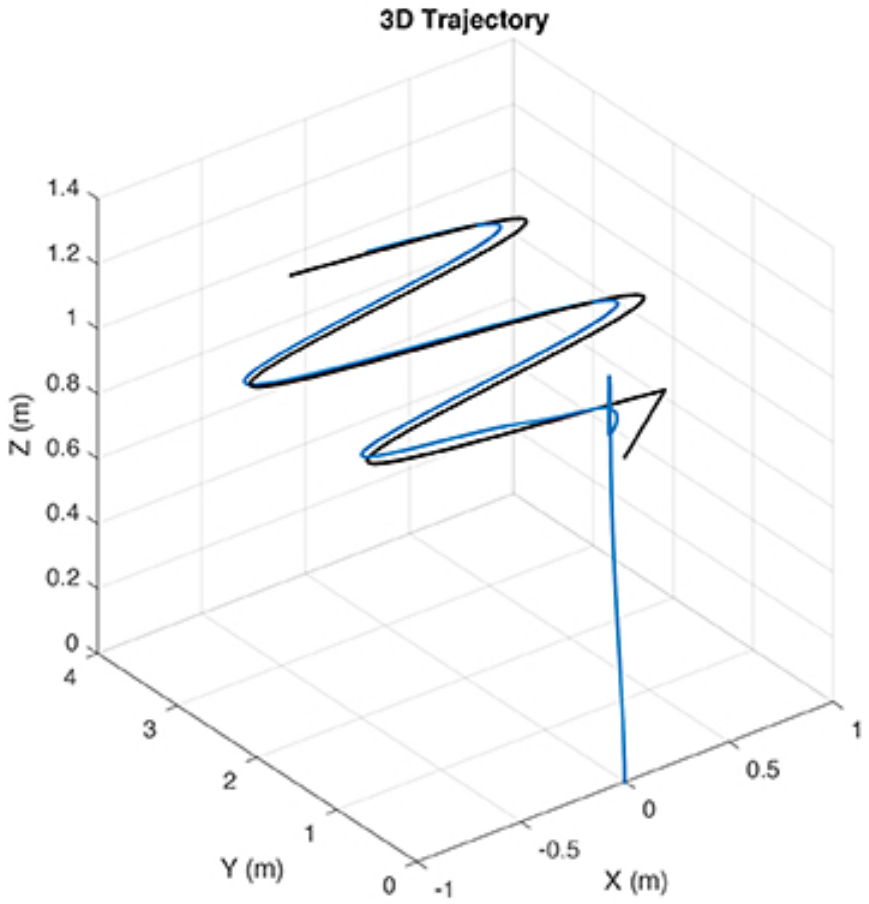
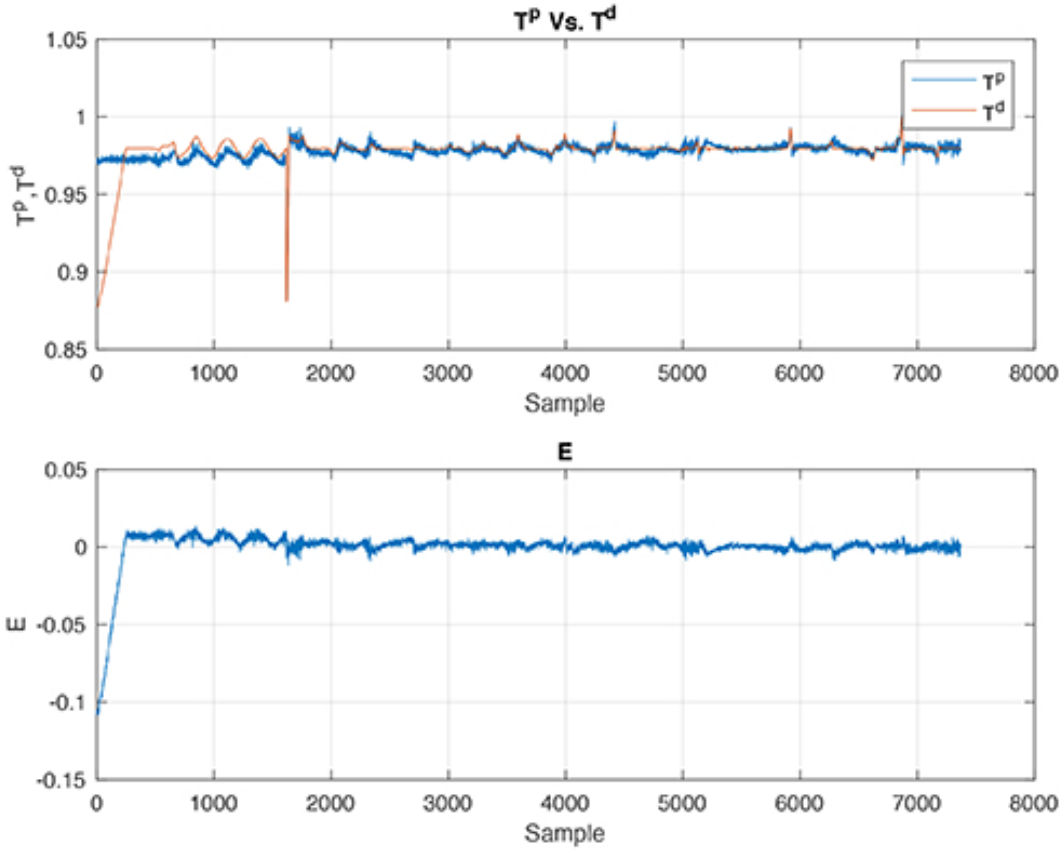
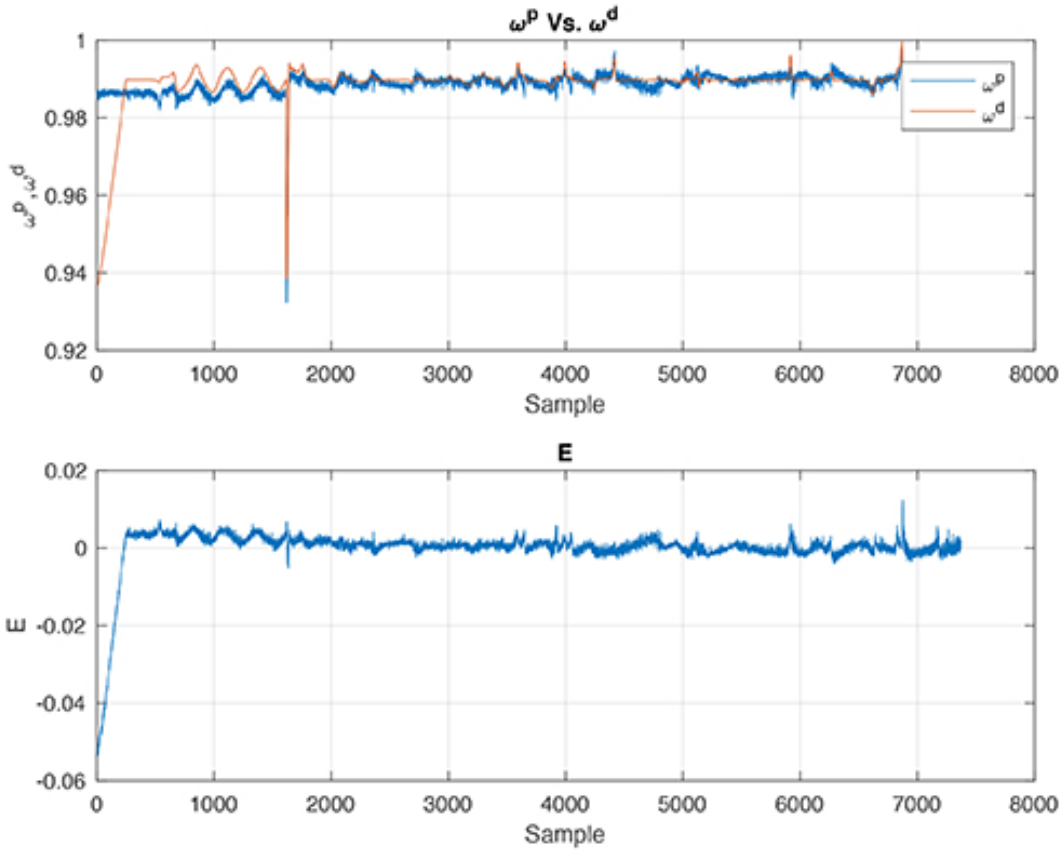


Fig. 3: Desired Trajectory: This figure shows the attitude control for the quadrotor after takeoff. The desired trajectory is shown in black, and the blue trajectory shows the quadrotor taking off before engaging attitude control.

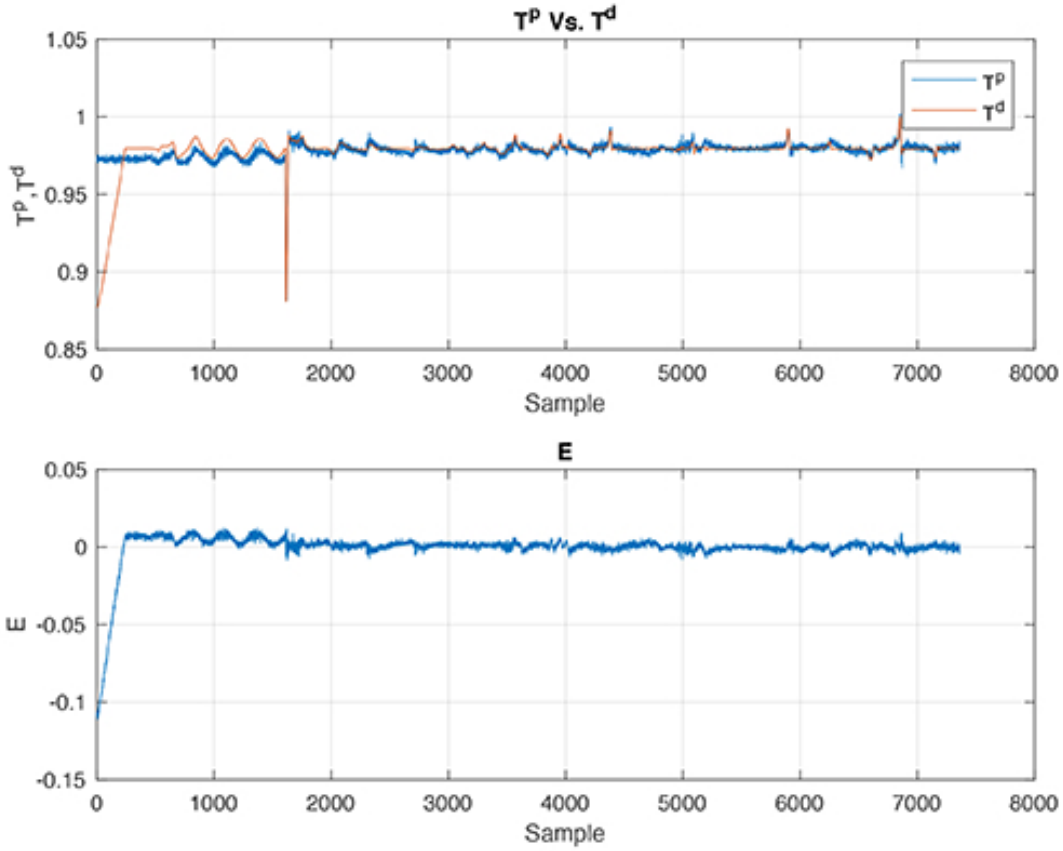


(a) Thrust Estimate

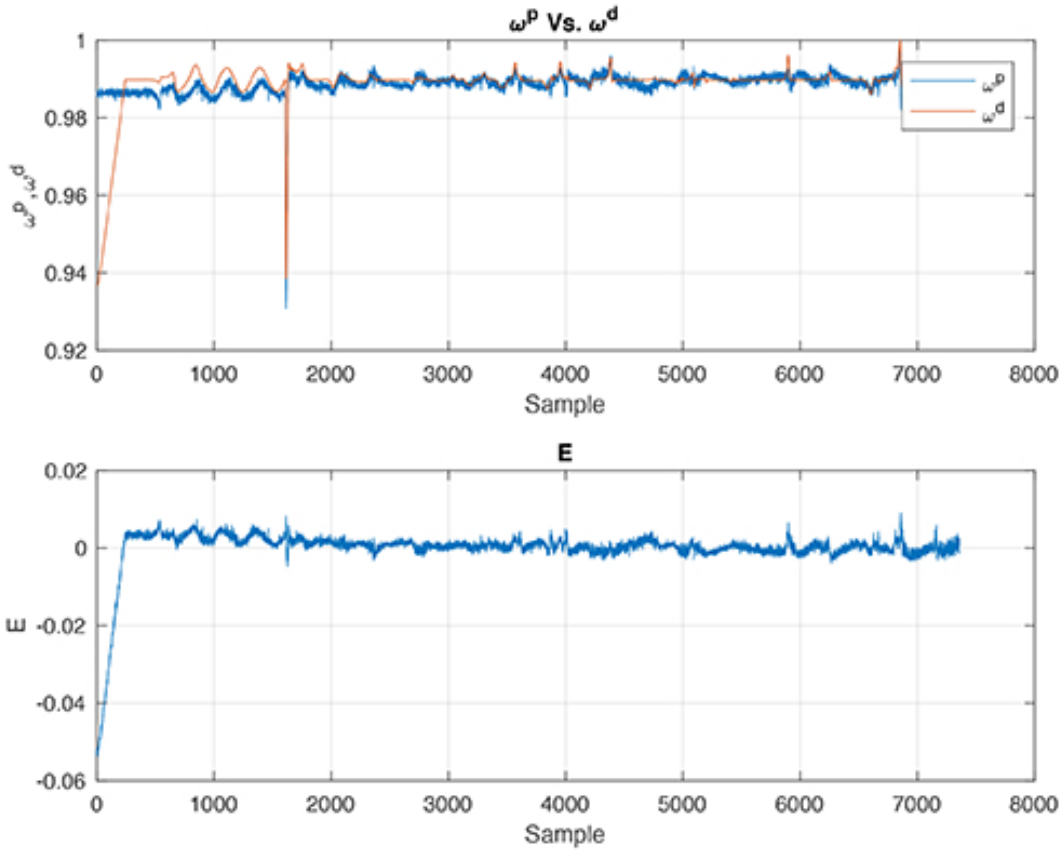


(b) Omega Estimate

Fig. 4: Blade 1

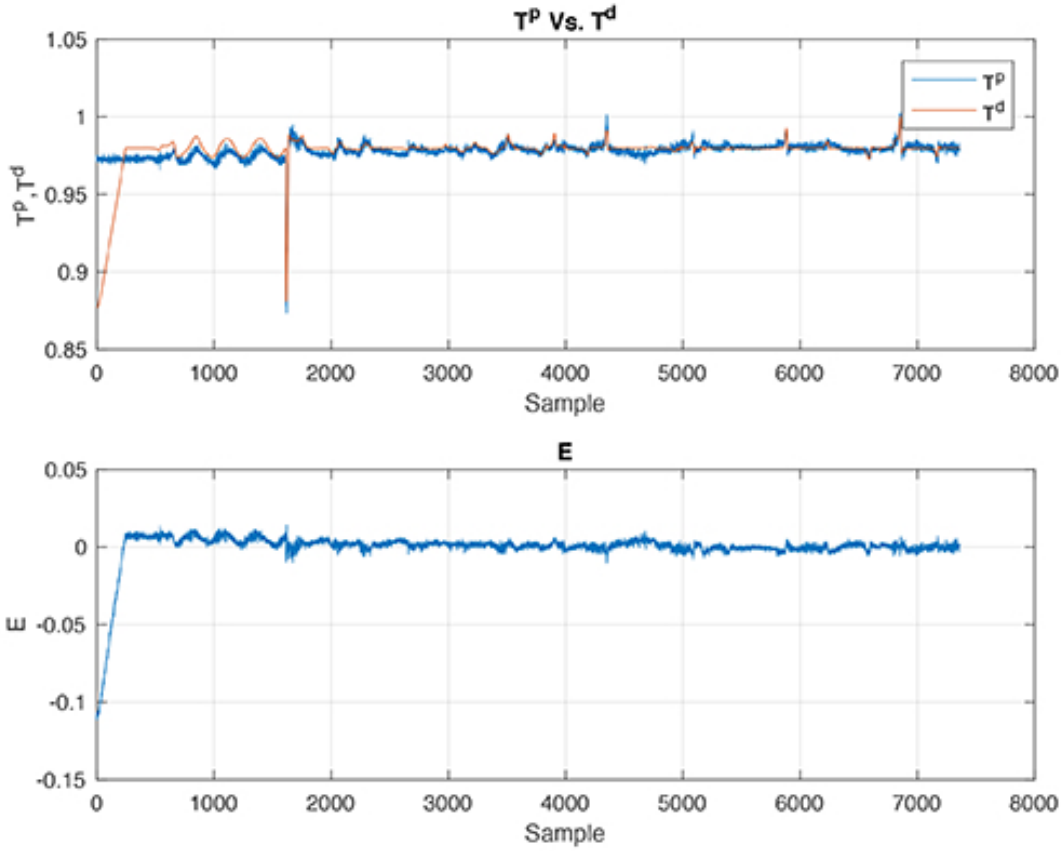


(a) Thrust Estimate

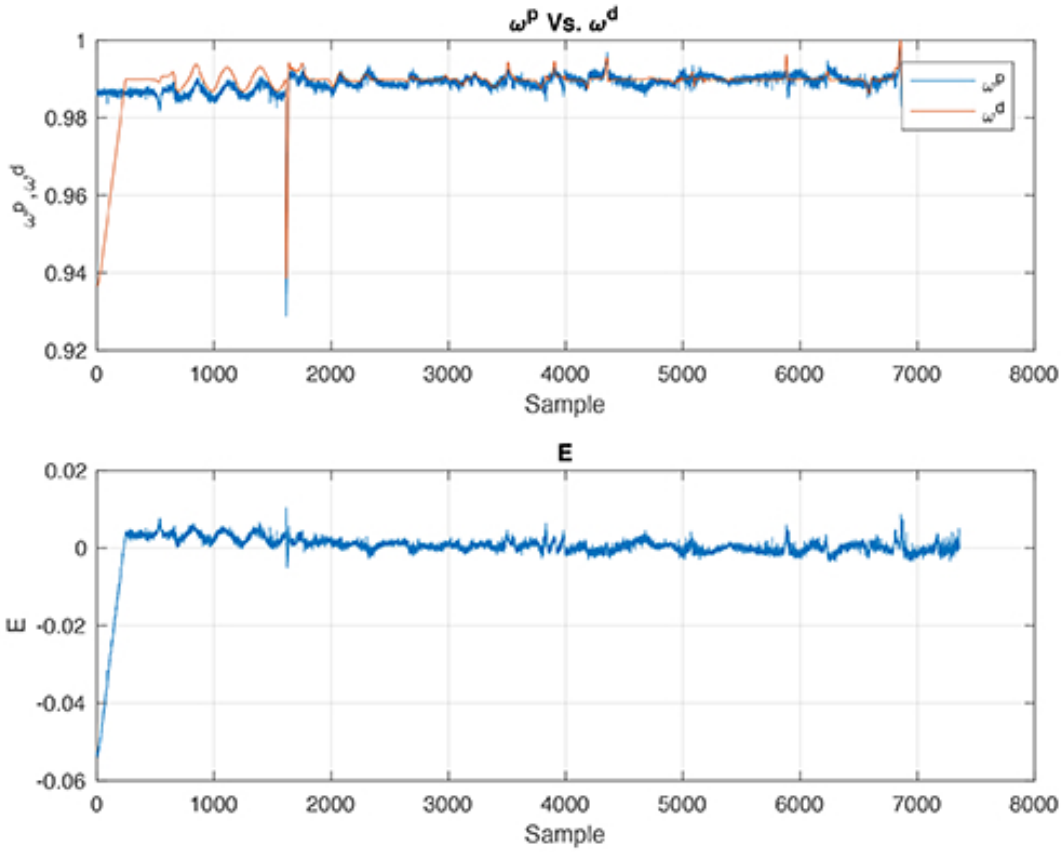


(b) Omega Estimate

Fig. 5: Blade 2

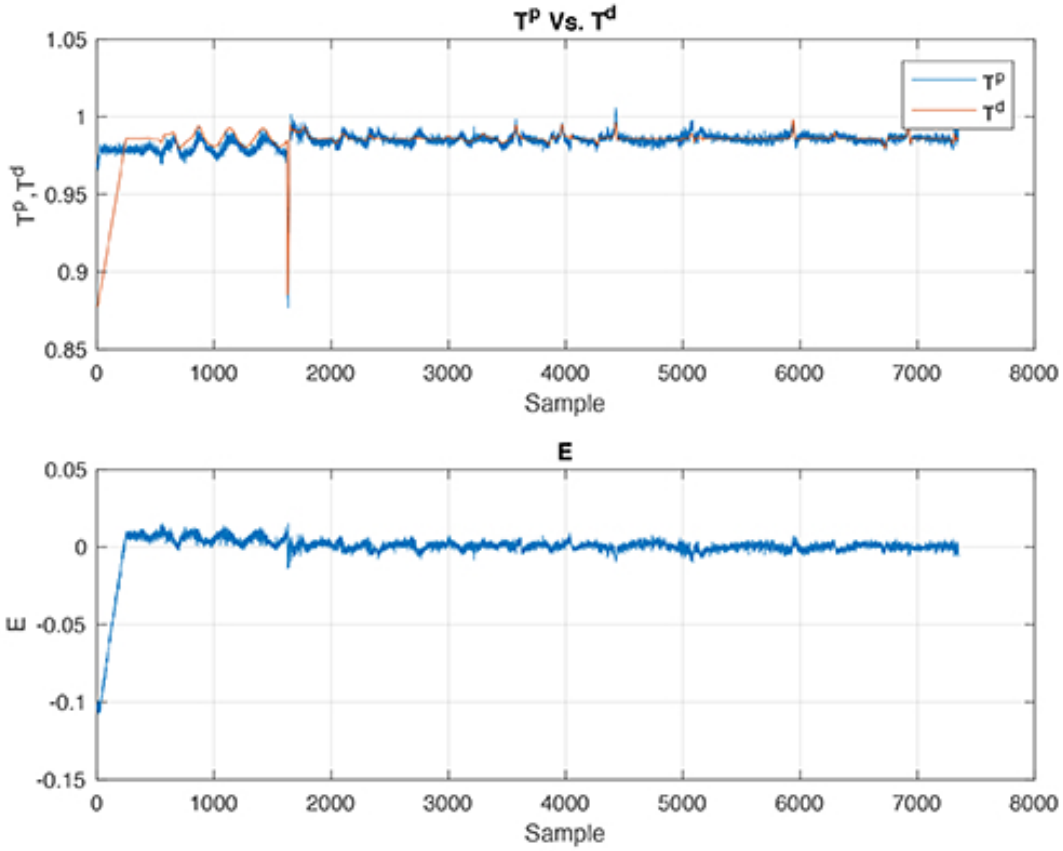


(a) Thrust Estimate

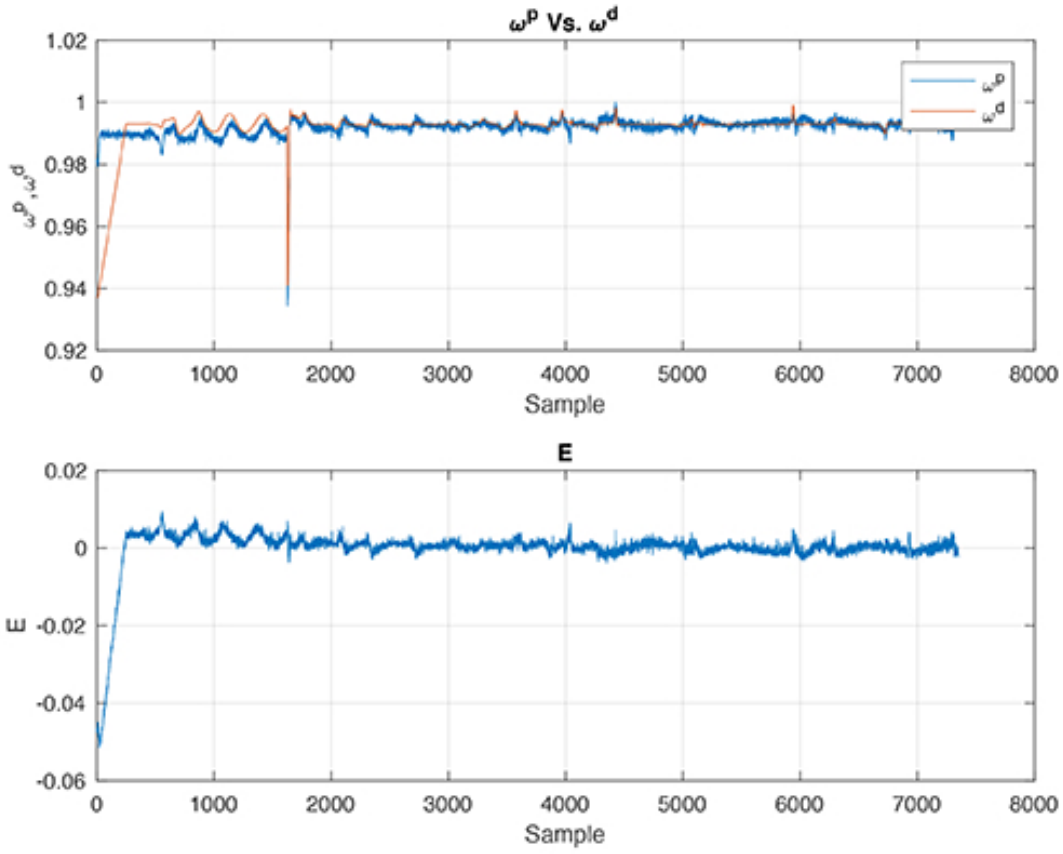


(b) Omega Estimate

Fig. 6: Blade 3



(a) Thrust Estimate



(b) Omega Estimate

Fig. 7: Blade 4

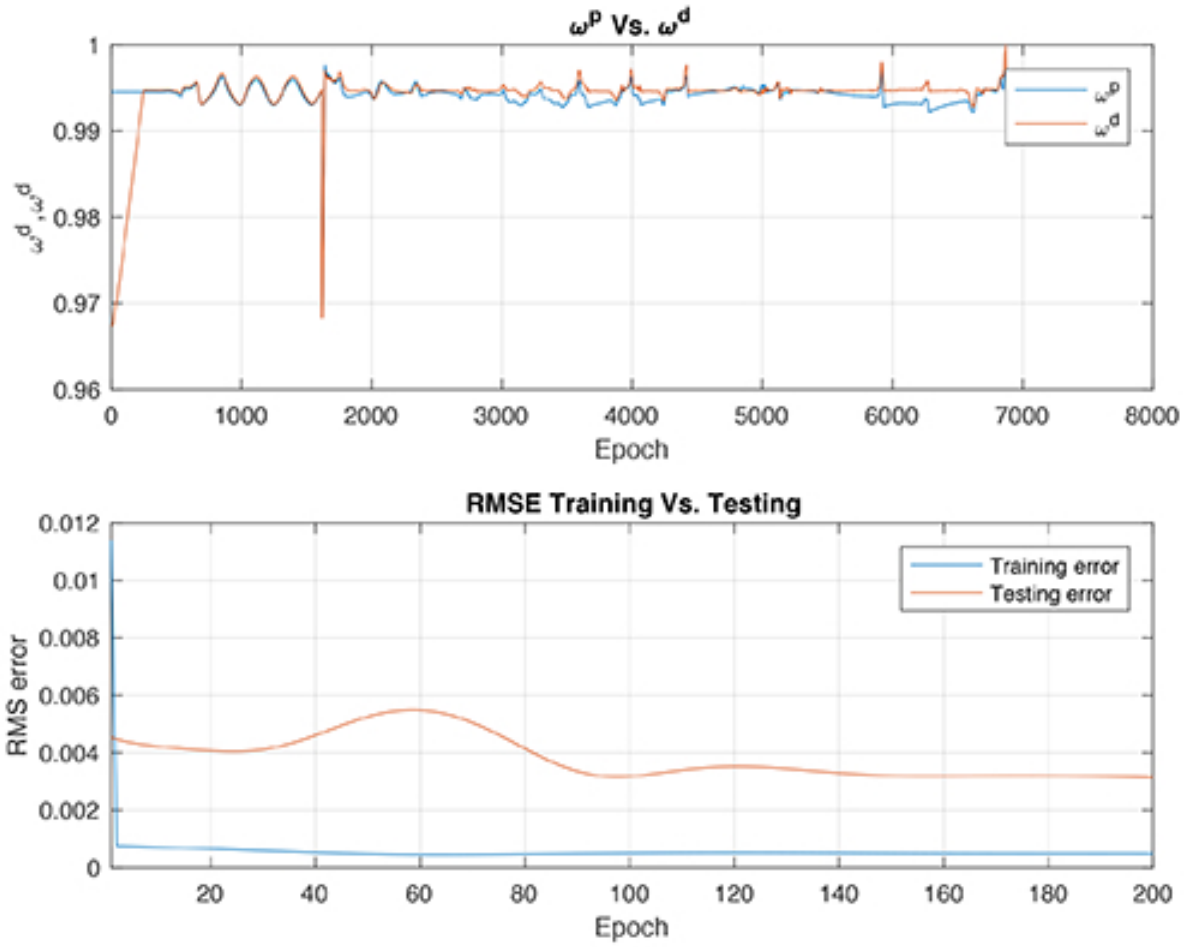


Fig. 8: Neural Network Approach: Bade 1

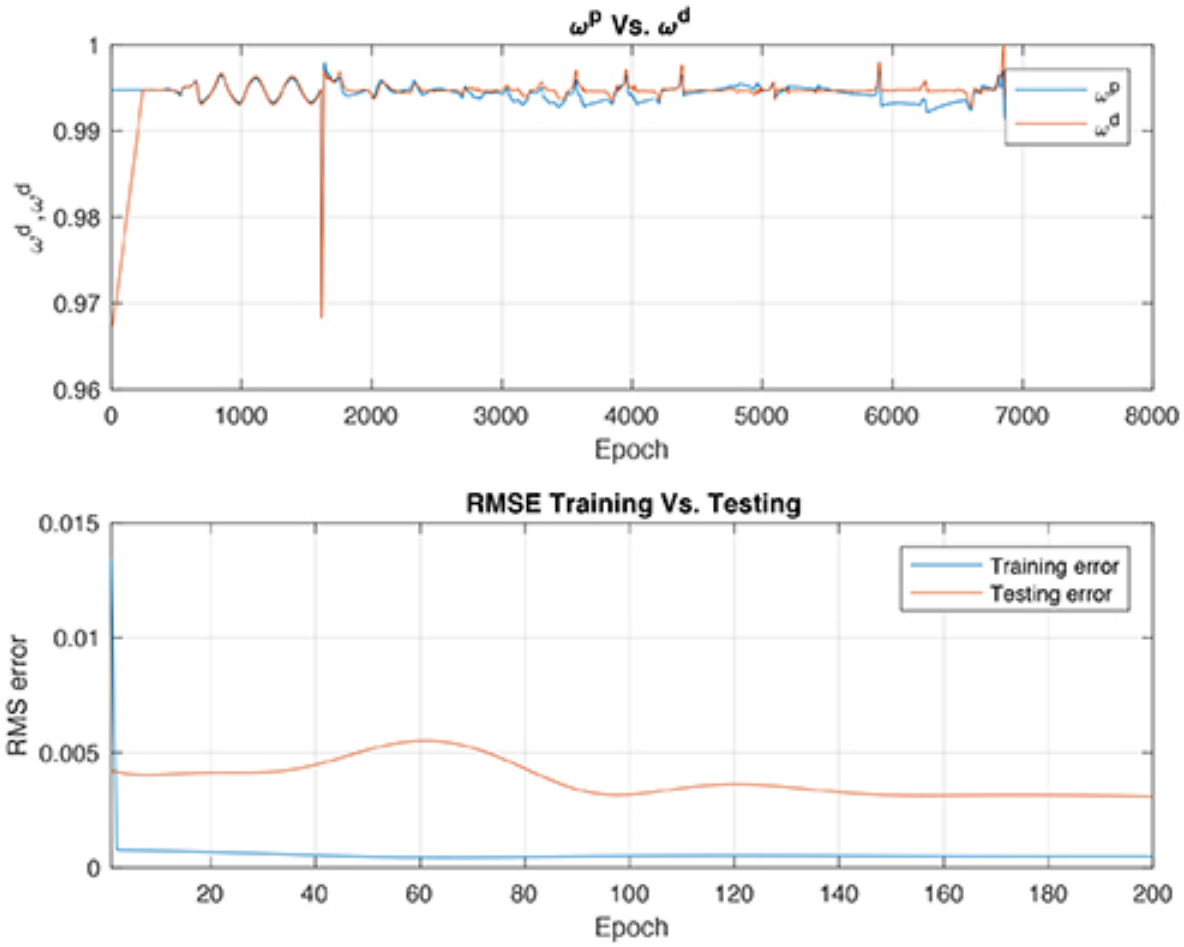


Fig. 9: Neural Network Approach: Bade 2

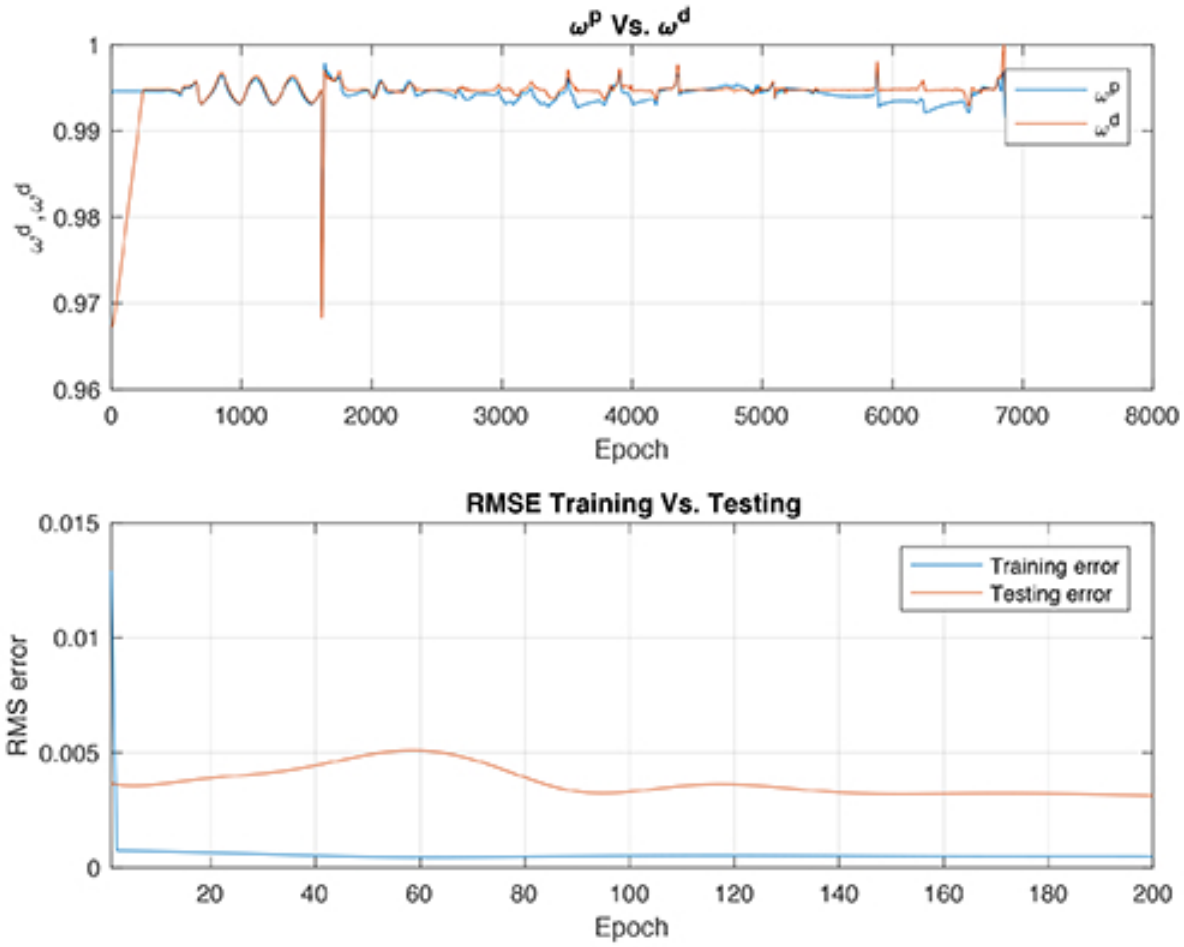


Fig. 10: Neural Network Approach: Bade 3

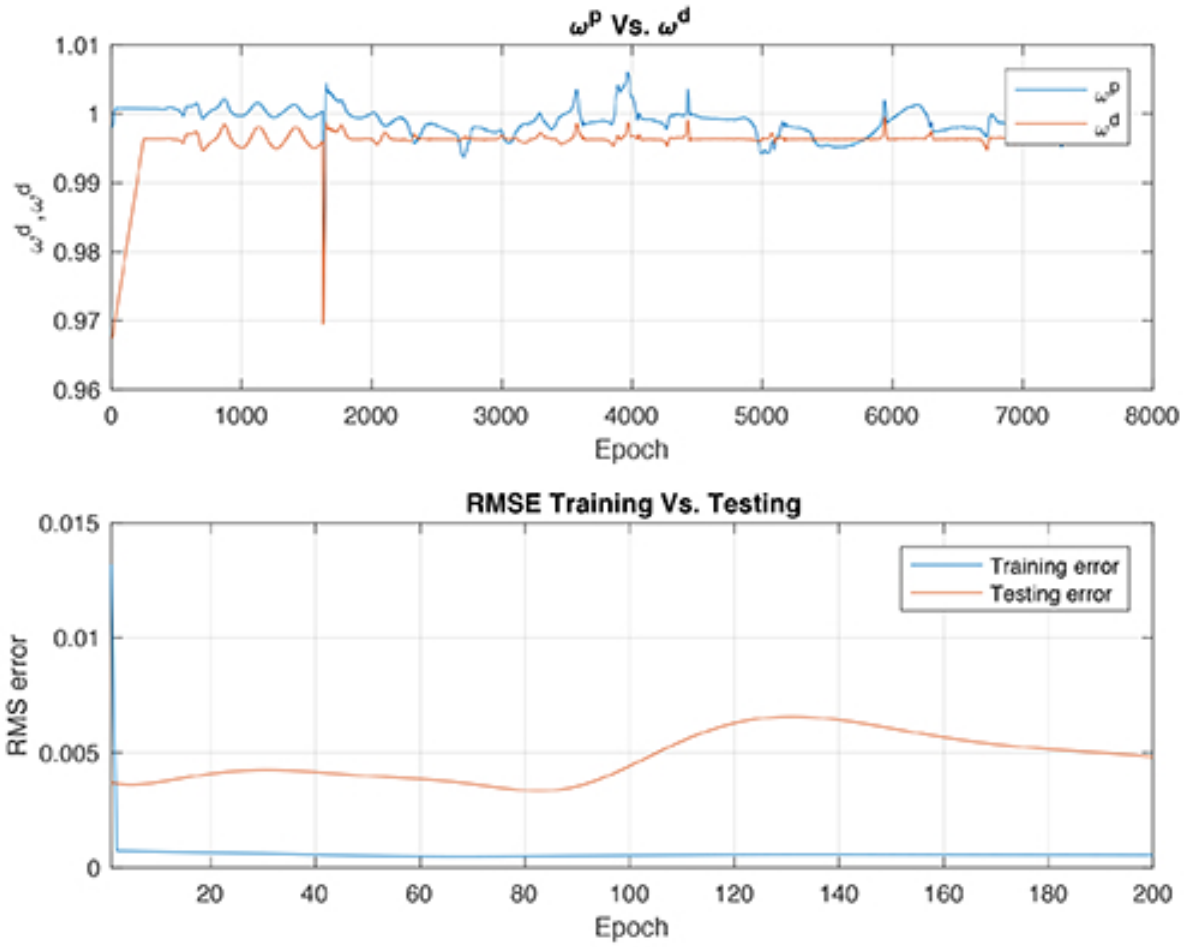


Fig. 11: Neural Network Approach: Bade 4

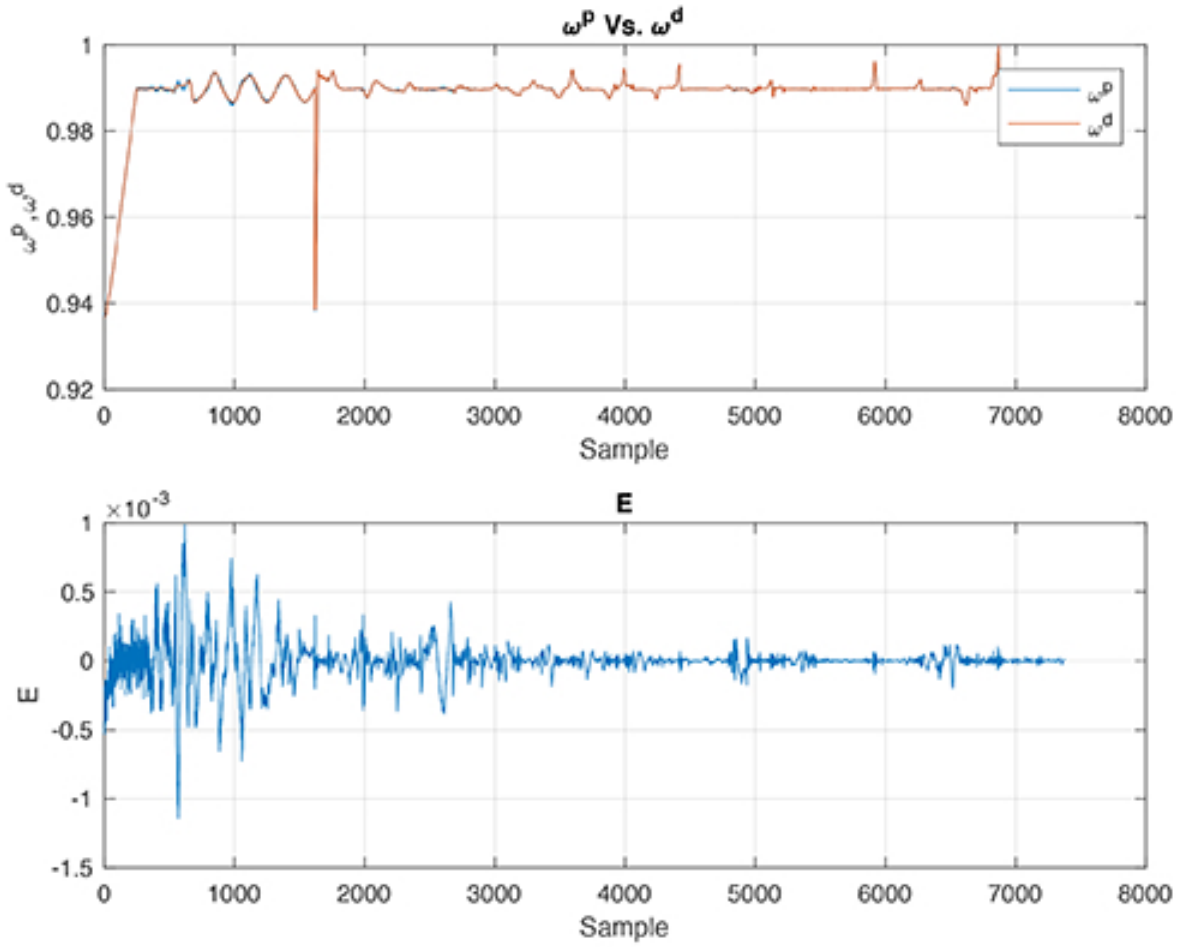


Fig. 12: Non-Sparse GP: Bade 1

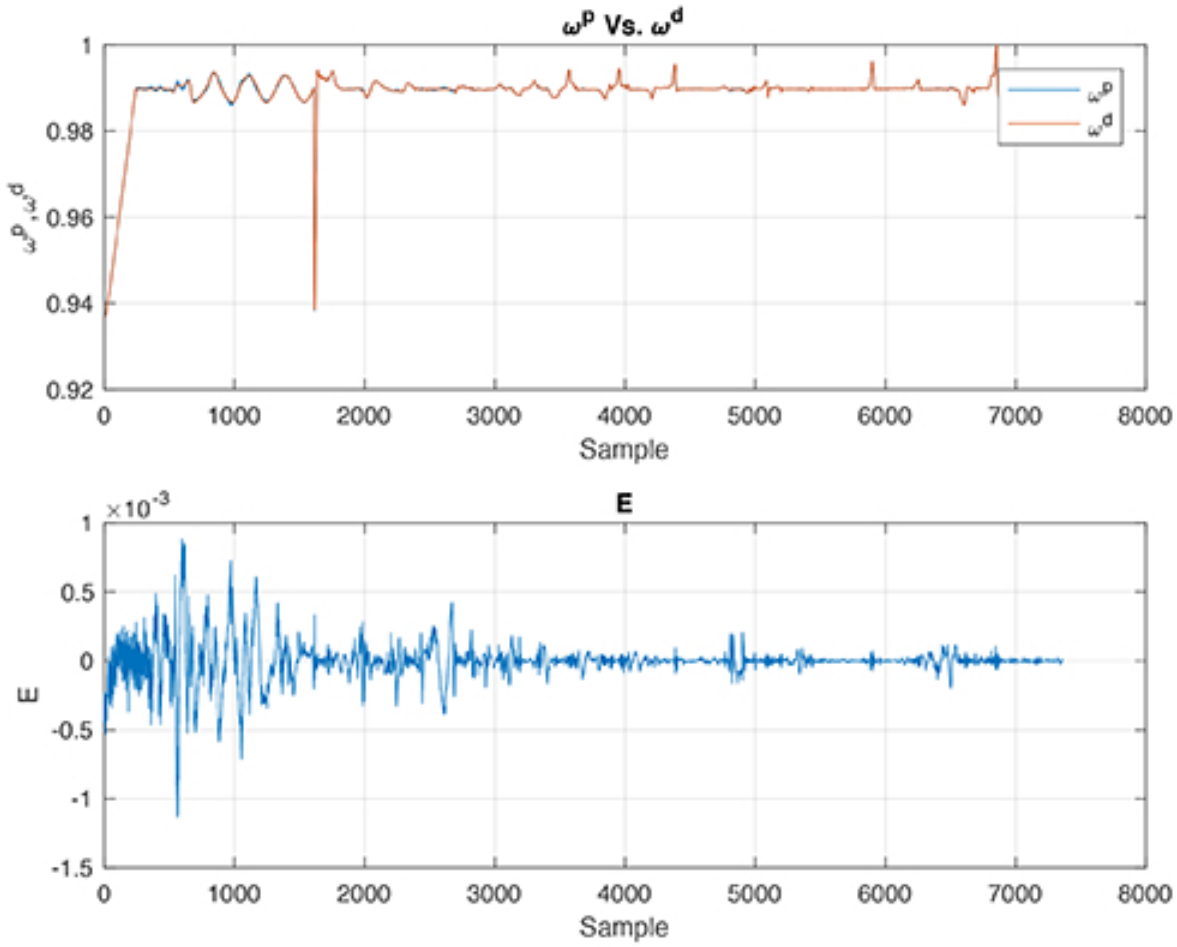


Fig. 13: Non-Sparse GP: Bade 2

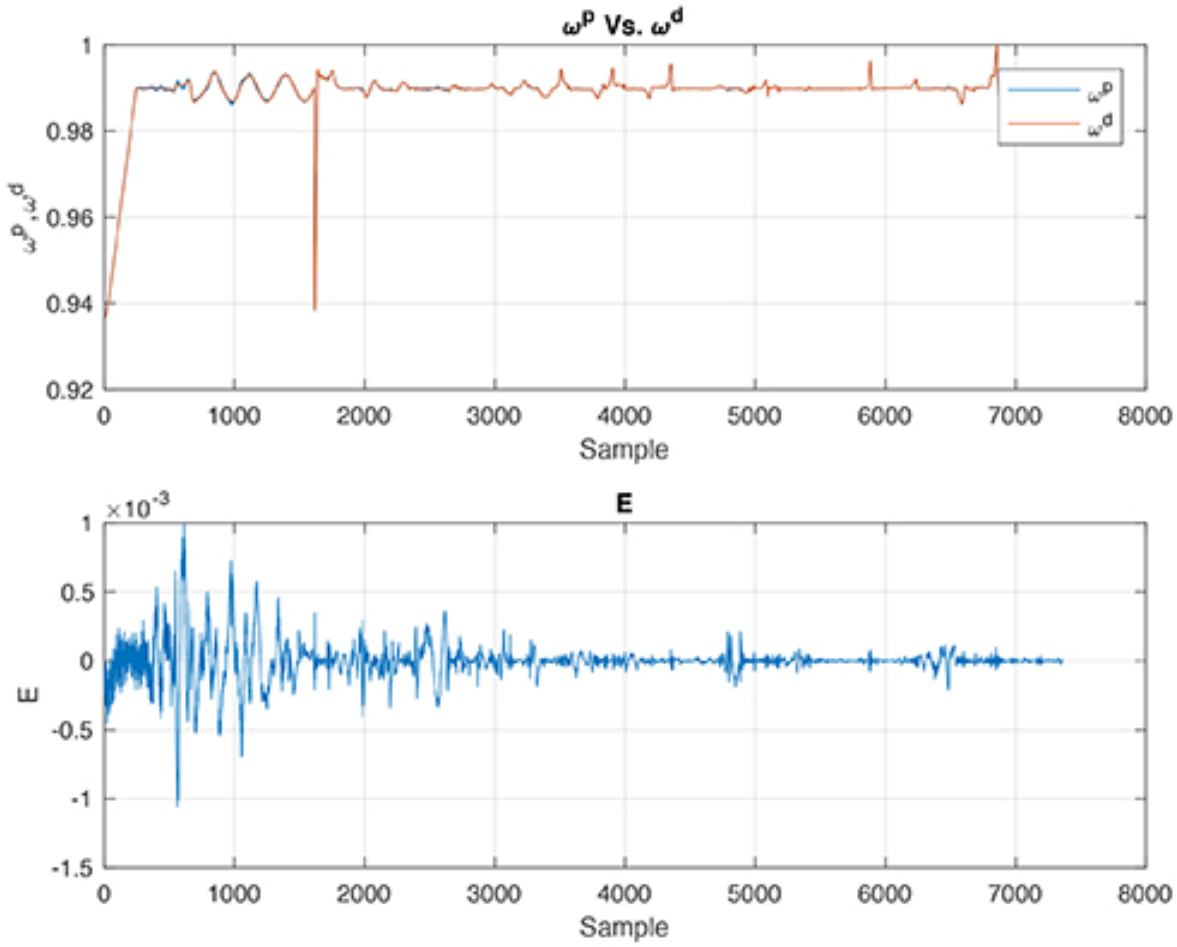


Fig. 14: Non-Sparse GP: Bade 3

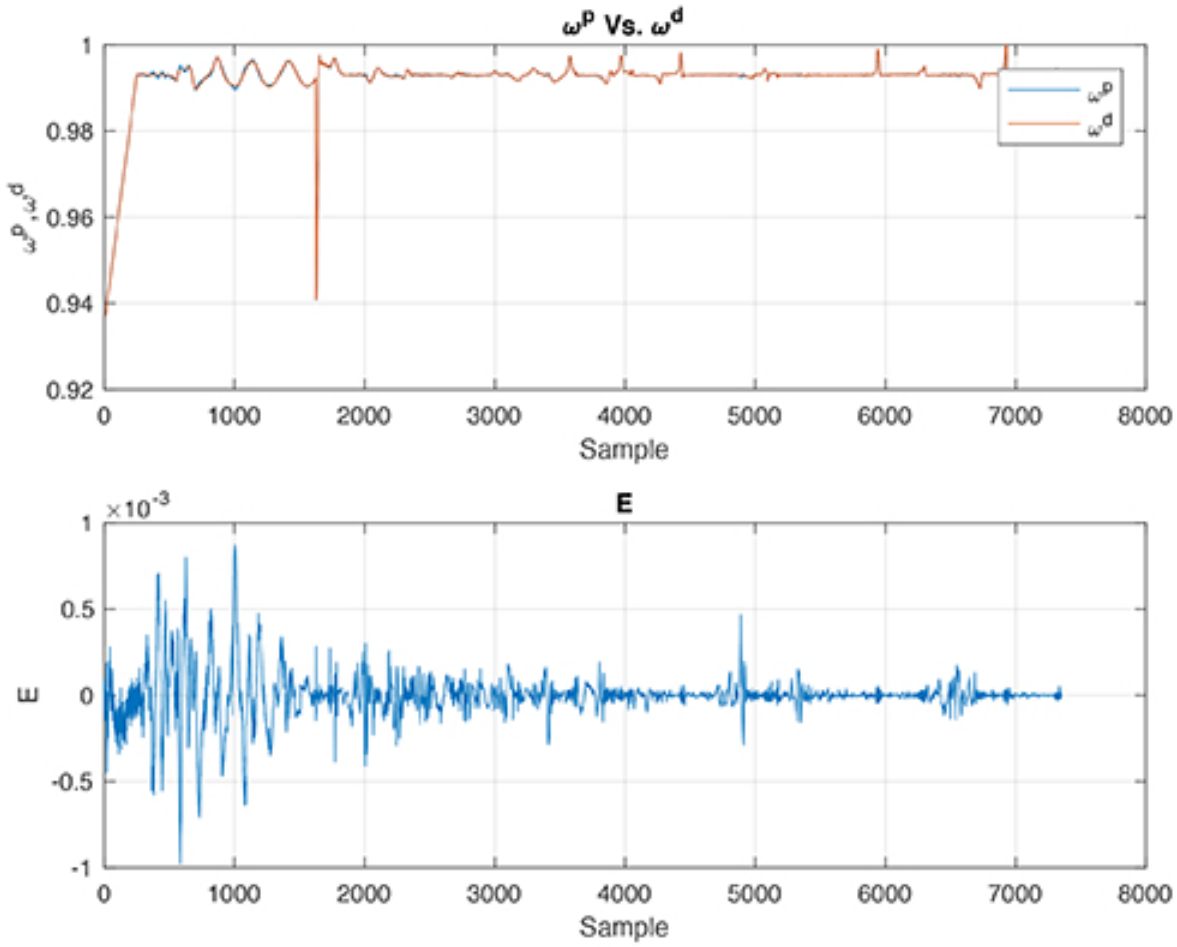


Fig. 15: Non-Sparse GP: Bade 4