# Adaptive Neural Network Quadrotor Trajectory Tracking Controller Tolerant to Propeller Damage

Mauro Villanueva Aguado*, Christophe De Wagter†, and Guido de Croon‡
Micro Air Vehicle Lab, Faculty of Aerospace Engineering,
Delft University of Technology, 2629 HS Delft, The Netherlands

http://www.imavs.org/

ABSTRACT

Accurate trajectory tracking with quadrotors is a challenging task that requires a trade-off between accuracy and complexity to run onboard. State-of-the-art adaptive controllers achieve impressive trajectory tracking results with slight performance degradation in varying winds or payloads but at the cost of computational complexity. This work proposes a lightweight combination of adaptive and neural control and shows its performance when flying with propeller damage. The neural architecture consists of offline learning of a condition-invariant representation of the aerodynamic forces through Deep Neural Networks. The second part consists of fast online adaptation using a composite adaptation law. We deploy this flight controller fully onboard the flight controller of the Parrot Bebop 1, showcasing its computational efficiency. The adaptive neural controller improves tracking performance by $\approx 60\%$ over the nonlinear baseline, with minimal performance degradation of just $\approx 20\%$ with increasing propeller damage.

## 1 INTRODUCTION

Unmanned Aerial Vehicles (UAVs) offer unmatched versatility & agility, with the potential to revolutionize a wide range of industries including cinematography, defense, agriculture, and logistics. Crucially UAVs require a trained operator to fly, increasing operational costs, and hindering the wider commoditization of UAVs. Autonomous UAVs take away the need for a trained operator. Autonomous UAVs have already been deployed in niche applications, from making history by executing the first controlled flight on another planet [1] to delivering crucial medical supplies to isolated hospitals in Africa [2].

One of these challenges is tracking trajectories accurately and thus safely in a wide range of conditions, which relies on aerodynamics modeling of the quadrotor. But these aerodynamic effects are difficult to model, "they consist of the propeller lift and drag which are dependent on the induced air stream velocity, fuselage drag, downwash and turbulent interactions between propellers and fuselage" [3]. This problem is further compounded in the case that the propellers are damaged which significantly changes the aerodynamic model of the quadrotor.

Conventional approaches to trajectory tracking use simple linear or quadratic [4] drag models to capture these aerodynamic effects. While simple yet effective at low speeds the accuracy of these models quickly degrades at higher speeds and cannot account for changes in the aerodynamic model. While high-fidelity aerodynamic models have been derived from Computational Fluid Dynamic (CFD) simulations [5], these simulations require platform-specific meshing of hundreds of millions of grid points and multiple days in large compute clusters to solve and are ultimately condensed into simplified models to be tractable onboard. Advances in Neural Networks (NN) and small powerful GPUs have enabled promising results at approximating these aerodynamic effects [6, 7, 8] while remaining computationally tractable to run onboard a quadrotor. NN provide a higher accuracy than the simple drag models. However, implementing NN in a drone controller while addressing the unpredictable nature of their output remains a largely unexplored challenge. A trade-off between model accuracy & complexity must be considered.

This paper is structured as follows. An overview of related work on quadrotor trajectory tracking control, adaptive control and adaptive neural control is presented in section 2. In section 3 a detailed methodology of the quadrotor model, adaptive neural controller and trajectory generation is provided. The experimental setup is explained in section 4. The main results are shown in section 5, including neural network training & validation loss, unmodeled force predictions, controller tracking performance comparison and trajectory plots. This is followed by discussing the results and providing recommendations in section 6. Lastly, the main conclusions of the adaptive neural controller are raised in section 7.

## 2 RELATED WORK

### 2.1 Quadrotor Trajectory Tracking Control

Quadrotors are inherently nonlinear and unstable platforms. Early work on quadrotor control achieved stable hover and near-hover flight using well-established control schemes such as Proportional-Integral-Derivative (PID) [9] or Linear-Quadratic Regulator (LQR) [10]. These controllers rely on the small-angle assumption to linearize the dynamics of the

*Email address: contact@maurovillanueva.eu
†Email address: c.dewagter@tudelft.nl
‡Email address: g.c.h.e.decroon@tudelft.nl

http://www.imavs.org/

system, thus they are most effective at low speeds. Despite this drawback cascaded PID control is the most common controller architecture in off-the-shelf flight controllers due to its simplicity, ease-of-tuning and sufficiently good performance.

As quadrotors have become more powerful and lightweight, capable of executing aggressive trajectories, nonlinear flight controllers have been proposed to cope with the non-linearities in the attitude dynamics. Nonlinear flight controllers such as Nonlinear Dynamic Inversion (NDI) [11] transform the nonlinear dynamics into a linear input-output map enabling the use of a linear control law. However, exact dynamic inversion suffers from a lack of robustness as it quite sensitive to sensor noise as well as modeling uncertainty [12]. Variants of NDI have been proposed to solve this problem, such as backstepping design [13] which recursively designs a controller, starting from a known stable inner system and progressively *backs-out* stabilizing the entire nonlinear system. A more recent variant known as incremental nonlinear dynamic inversion (INDI) [14] has been developed, which improves robustness by gradually applying control inputs based on inertial measurements.

State-of-the-art trajectory tracking algorithms can be categorized into predictive or non-predictive methods. Non-predictive methods track a single reference step while predictive methods encode several future timesteps into the control command. Differential-Flatness-Based Control (DFBC), a non-predictive method, takes advantage of the fact that quadrotors are differentially flat systems [15] allowing for the reformulation of the trajectory tracking problem into a state tracking problem. DFBCs are combined with an outer loop INDI controller and aerodynamic model to achieve an impressive tracking performance at high-speeds [16, 17], with a Position Root-Mean-Squared Error (RMSE) of as little as 12.2 $cm$ with a top speed of 20 $m/s$ [17]. However, [17] relies on optical encoders for direct motor speed feedback or in the case [16], is limited to x-y plane trajectories and uses a very high-frequency attitude controller running at 4 $kHz$. Most importantly, DFBC relies on the quadrotor model to generate the reference states and thus is susceptible to modeling mismatch, reducing the thrust coefficient by 30% will lead to significant performance degradation or even crashing, with position RMSE degrading from 8.5 $cm$ to 100.4 $cm$ [18].

Nonlinear Model Predictive Control (NMPC) is the most prevalent predictive method. NMPCs generate motor commands in a receding horizon fashion, solving the constrained nonlinear optimization problem over the predicted time horizon and minimizing tracking error. Similarly, the performance of NMPCs is further improved by adding an outer loop INDI controller and aerodynamic model. NMPCs can minimize the tracking error across multiple future time-steps whereas DFBCs are too short-sighted only considering one reference point. This allows NMPCs to outperform DFBCs at tracking trajectories at high speeds, especially for dynamically infeasible trajectories. NMPCs achieve a state-of-the-art position RMSE of 10.2 $cm$ with a maximum speed of 20 $m/s$ [18]. However, solving this nonlinear optimization problem over several future time steps requires significant computational resources, in the order of 100 times greater than DFBC. Despite advances in powerful embedded computers and nonlinear solvers deploying NMPC onboard is still a challenge, requiring an NVIDIA Jetson TX2 to run the algorithm onboard at 100 $Hz$ in [18] adding significant weight and power consumption.

### 2.2 *Adaptive Control*

Adaptive control of systems with parametric uncertainty has been extensively researched. Adaptive controllers are often an augmentation to existing controllers rather than a standalone controller. In the field of UAVs adaptive controllers can be categorized into Model Identification Adaptive Controllers (MIACs) or Model Reference Adaptive Controllers (MRACs). MRACs use an adaptive controller, typically an $\mathcal{L}_1$ adaptive controller, to drive the system towards a desired reference model behavior. MRACs have been successfully deployed in quadrotors, demonstrating slight trajectory tracking degradation with unmodeled weights attached [19, 20] or loss-of-thrust [21]. Albeit, these results are achieved tracking simple circular trajectories not exceeding 2 $m/s$ with a ground station in-the-loop [19], require powerful embedded computers [20] or have overall poor tracking performance [21].

MIACs perform System Identification (SI) online to estimate the values of unknown linear coefficients which are then mixed with known basis functions. The selection of basis functions may be challenging, but they should reflect important features of the underlying dynamics. Physics-based modeling is typically used to design these basis functions [22], however, this requires extensive knowledge of the system and suffers from convergence problems when there is a lack of excitation. Alternatively, random Fourier features can be used as basis functions as they can capture all underlying dynamics given a sufficient number of features. However, the high-dimensional feature space may not be an efficient representation as features are redundant or irrelevant and lack the representation power of Deep Neural Networks (DNNs). Nevertheless, random Fourier features are used in [23] to learn an acceleration error model of a quadrotor with a drag plate attached in windy conditions improving on the performance of an $\mathbf{L}_1$ adaptive controller.

### 2.3 *Adaptive Neural Control*

The ability of NNs to accurately and computationally efficiently approximate nonlinear functions has drawn interest in the field of adaptive control. Early work focused on solving the identification problem of nonlinear systems using NNs to create a reference model [24, 25] whose weights are updated online based on the error between the system and model output. These networks were shallow and lacked pretraining, limiting their performance and requiring the initial guess of

http://www.imavs.org/

network weights to be close to the correct guess to ensure stability. More recently, a quadrotor has performed an impressive flip in wind using an NN to predict the unknown forces [7], the weights are updated based on the position & velocity error. Albeit, the network is still limited to a simple 3-layer [6,3,3] neuron architecture lacking pretraining.

Lately, meta-learning has drawn attention as a scheme that "learns-to-learn" efficient models of systems from data gathered in varying conditions. The learned model is capable of fast adaptation to highly dynamic environments. Meta-Learning algorithms typically can be decomposed into two phases: offline learning and online adaptation. In the offline learning phase, the goal is to learn a model from data collected in various environments that capture common features shared across all environments. In the online adaptation phase, the goal is to adapt this model to the current environment based on the data available. A naive approach to online adaptation would adapt the entire network online, however this is a computationally expensive task severely limiting the depth of the network and lacks stability guarantees to unpredictable outputs. A more sophisticated approach augments the NN output with latent variables identified online which represent the unknown environmental factors allowing for rapid adaptation with little computational burden. Meta-Learning adaptive control has achieved successful adaptive control of quadrotors carrying suspended payloads [26] and strong winds [27]. Of particular interest are the results achieved in [27] whose "Neural-Fly" controller demonstrates state-of-the-art tracking performance in strong & varying wind conditions achieving a position RMSE of $9.4cm$ at wind speeds of $12.1\ m/s$ with little computational cost.

## 3 METHODOLOGY

In this paper vectors are denoted in lowercase bold $\mathbf{p}$ and matrices in uppercase bold $\mathbf{I}$, otherwise, they are scalars. Two right-handed coordinate frames are used in this paper shown in Figure 1, the body frame: $\mathcal{F}_B : \{\boldsymbol{x}_B, \boldsymbol{y}_B, \boldsymbol{z}_B\}$ located at the center of mass of the quadrotor with $\boldsymbol{x}_B$ pointing forward and $\boldsymbol{z}_B$ aligned with the collective thrust direction and the inertial world frame: $\mathcal{F}_W : \{\boldsymbol{x}_W, \boldsymbol{y}_W, \boldsymbol{z}_W\}$ with $\boldsymbol{z}_W$ pointing in the opposite direction of gravity. Vectors with the superscript $\square^B$ are expressed in the body frame and those without one are expressed in the world frame. The rotation from inertial frame to body frame is represented by the rotational matrix $\boldsymbol{R} = [\boldsymbol{x}_B, \boldsymbol{y}_B, \boldsymbol{z}_B] \in \mathrm{SO}(3)$.

### 3.1 *Quadrotor Model*

The quadrotor model is based on the model provided in [17], modeled as a 6-degree-of-freedom rigid body with 4 inputs corresponding to the commanded rpms of the motors. The translational dynamics is given by:

$$\boldsymbol{a}^I = \frac{\boldsymbol{f}_t + \boldsymbol{f}_a + \boldsymbol{f}_{res}}{m} + \boldsymbol{g} \tag{1}$$

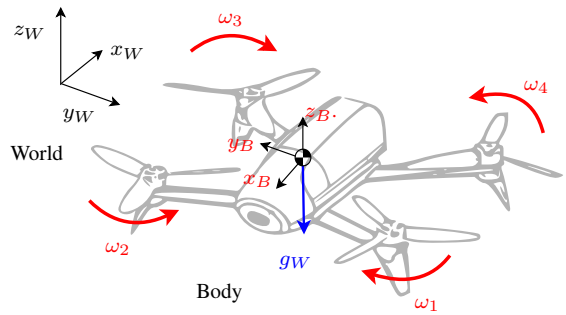



Figure 1: Coordinate System Definitions

where $\boldsymbol{a}$ is the acceleration of the quadrotor's center of gravity; $\boldsymbol{f}_t$ is the collective thrust; $m$ is the quadrotor mass; $\boldsymbol{f}_a$ represents the aerodynamic model forces; $\boldsymbol{f}_{res}$ captures the remaining unmodeled forces; $\boldsymbol{g} = \left[0, 0, -9.81\ \mathrm{ms}^{-2}\right]^T$ is the gravity vector. The rotational kinematics and dynamic equations are expressed as:

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \boldsymbol{\Omega}^B \tag{2}$$

$$\dot{\boldsymbol{\Omega}}^B = \boldsymbol{I}^{-1}\left(\boldsymbol{\tau}^B + \boldsymbol{\tau}^B_{ext} - \boldsymbol{\Omega}^B \times \boldsymbol{I}\boldsymbol{\Omega}^B\right) \tag{3}$$

where $\boldsymbol{\Omega}^B$ is the body frame angular velocity vector; $\boldsymbol{I} \in \mathbb{R}^{3\times3}$ is the quadrotor diagonal moment of inertia matrix; $\boldsymbol{\tau}$ are the modeled body torques and $\boldsymbol{\tau}_{ext}$ accounts for the any unmodeled disturbance moments. The symbol $\otimes$ represents the Hamilton quaternion product.

#### 3.1.1 Thrust and Torque Model

The thrust and torque acting on the quadrotor are modeled in Equation 4 & Equation 5 respectively, we assume stiff propellers with no rotor drag and neglect moments caused by aerodynamic effects, the angular acceleration of rotors and gyroscopic effects.

$$\boldsymbol{f}_t^B = \left[0, 0, c_t \sum_{i=1}^{4} \omega_i^2\right]^T \tag{4}$$

$$\boldsymbol{\tau}^B = \begin{bmatrix} l_y c_t & -l_y c_t & -l_y c_t & l_y c_t \\ l_x c_t & l_x c_t & -l_x c_t & -l_x c_t \\ c_\tau & -c_\tau & c_\tau & -c_\tau \end{bmatrix} \boldsymbol{\omega}^2 \tag{5}$$

where $c_t$ is the propeller thrust coefficient and $c_\tau$ the propeller torque coefficient; $l_x$ and $l_y$ are the moment arms shown in Figure 1.

#### 3.1.2 Aerodynamic Force Model

The aerodynamic model from [4] captures the major aerodynamic effects in a computationally efficient manner.

http://www.imavs.org/

$$\boldsymbol{f}_a^B = -\sum_{i=1}^{4} \omega_i \begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix} \odot \boldsymbol{v}^B + \begin{bmatrix} 0 \\ 0 \\ k_h \left( {v_x^B}^2 + {v_y^B}^2 \right) \end{bmatrix} \quad (6)$$

where $\boldsymbol{v}^B = \boldsymbol{R}(\boldsymbol{q})^T \boldsymbol{v}$ is the body frame velocity vector and $k_x, k_y, k_z, k_h$ are the aerodynamic coefficients identified from flight data in [28]. The symbol $\odot$ represents vector element-wise multiplication.

### 3.2 Adaptive Neural Controller

The adaptive neural controller design follows the "Neural-Fly" controller architecture closely in [27]. Neural-Fly is a data-driven trajectory tracking controller that uses a learning-based approach to achieve fast & accurate online adaption by incorporating pre-trained representations using deep learning. The controller has two main components: offline meta-learning and online adaptive control. These two components together build a model of the unmodeled forces acting on a quadrotor of the form:

$$\boldsymbol{f_{res}}(\boldsymbol{x}, \boldsymbol{k}) \approx \phi(\boldsymbol{x}) \boldsymbol{A}(\boldsymbol{k}) \quad (7)$$

where $\boldsymbol{\phi}$ is a representation function that maps the unmodeled forces dependent on the quadrotors state $\boldsymbol{x}$ to a shared representation across all sampled environments. $\boldsymbol{A}$ is a set of linear coefficients that adapt to the current quadrotor environment $k$. The offline training phase consists of training $\boldsymbol{\phi}$, a DNN using Domain Adversarial Meta-Learning (DAIML) to learn a propeller damage invariant representation of the unmodeled forces. The online adaptation phase aims to adapt the linear coefficients in $\boldsymbol{A}$ to the current propeller damage condition using a composite adaptive controller while maintaining stability and robustness.

Combining the translational dynamics of a quadrotor in Equation 1 with the estimate of the unmodeled force in Equation 7 we arrive at the control law of the adaptive neural controller:

$$\boldsymbol{u}_f = \underbrace{m\boldsymbol{a}_r^B + \boldsymbol{R}\boldsymbol{f}_a^B + m\boldsymbol{g}^B}_{\text{nominal model feedforward}} - \underbrace{\boldsymbol{K}_p\boldsymbol{e}_p - \boldsymbol{K}_v\boldsymbol{e}_v}_{\text{PD feedback}} - \underbrace{\boldsymbol{\phi}(\boldsymbol{x})\hat{\boldsymbol{A}}}_{\text{adaptation}} \quad (8)$$

where $\boldsymbol{e}_p = p - p_r$ is the position tracking error, the subscript $\square_r$ denotes the reference trajectory, i.e. $a_r$ is the reference acceleration of the desired trajectory; $\boldsymbol{e}_v = v - v_r$ is the velocity tracking term; $\boldsymbol{K}_p$ and $\boldsymbol{K}_v$ are positive definite control gain matrices for position and velocity respectively. The output of Equation 8 is the control law $u_f$, a vector of commanded forces from which the corresponding attitude and thrust is obtained through inverse kinematics.

The adaptive neural controller replaces the position control loop, and the output attitude and thrust commands are sent to the inner attitude control loop and thrust mixer respectively. In contrast to a traditional PID controller, the adaptive neural controller also includes nominal model feed-forward terms to account for the known aerodynamic effects and desired acceleration, improving tracking of the reference trajectory. The architecture of the adaptive neural controller is shown in Figure 2.

This paper builds upon the work presented in [27] through the following contributions: (I) validation of the "Neural-Fly" adaptive neural controller architecture (II) adaptation to novel propeller damage conditions (III) analysis of the unmodeled force prediction performance on training, testing and flight data (IV) sensitivity analysis of the neural network architecture to hyperparameters (VI) implementation of the online adaptation phase onboard the flight controller in $C$.

#### 3.2.1 Offline Training

The goal of the offline training phase is to learn a representation function $\boldsymbol{\phi}(\boldsymbol{x})$ such that for any condition $k$ a latent variable $\boldsymbol{A}$ exists such that $\boldsymbol{\phi}(\boldsymbol{x})\boldsymbol{A}(k)$ approximates $f_{res}(\boldsymbol{x}, k)$ well. A DNN is used to learn this representation function $\boldsymbol{\phi}$, taking advantage of the representation power of DNNs to accurately approximate any nonlinear function given a sufficient amount of neurons and training data. The optimal representation $\boldsymbol{\phi}^*$ is typically obtained by minimizing the loss:

$$\min_{\boldsymbol{\phi}, \boldsymbol{A}^1, \cdots, \boldsymbol{A}^K} \sum_{k=1}^{K} \sum_{i=1}^{N^k} \left\| \boldsymbol{y}_i^k - \boldsymbol{\phi}(\boldsymbol{x}_i^k) \boldsymbol{A}^k \right\| \quad (9)$$

where $K$ is the total amount of conditions sampled and $N_k$ is the total amount of data-points collected with condition $k$. It is the Euclidean norm of the error between the measured unmodeled forces $\boldsymbol{y}$ and the network output $\boldsymbol{\phi}(x)\boldsymbol{A}$ across all sampled conditions and data points.

Note that the minimization problem also involves the latent variables $\boldsymbol{A}^1, \cdots, \boldsymbol{A}^K$, thus back-propagation is also performed through $\boldsymbol{A}$ to ensure that the optimal latent variables $\boldsymbol{A}^{k^*}$ are found.

However, the distribution of the quadrotor states $\boldsymbol{x}$ will vary depending on the propeller damage condition $k$ flown. For example, propeller $rpm$ values will increase as propeller damage worsens to cope with the loss of lift. This inherent domain shift in $x$ caused by a shift in $k$ may lead to overfitting of the network $\boldsymbol{\phi}$. The network $\boldsymbol{\phi}$ could learn the change in the unmodeled forces due to $k$ via the change in the distribution of $x$, instead of learning a propeller damage invariant representation. To solve this domain shift problem an adversarial loss is used:

$$\max_{h} \min_{\boldsymbol{\phi}, \boldsymbol{A}^1, \cdots, \boldsymbol{A}^K} \sum_{k=1}^{K} \sum_{i=1}^{N^K} \left( \left\| \boldsymbol{y}_i^k - \boldsymbol{\phi}(\boldsymbol{x}_i^k) \boldsymbol{A}^k \right\| - \alpha \cdot \mathcal{L}_{\boldsymbol{h}} \left( \boldsymbol{h}\left(\boldsymbol{\phi}(\boldsymbol{x}_i^k)\right), k \right) \right) \quad (10)$$

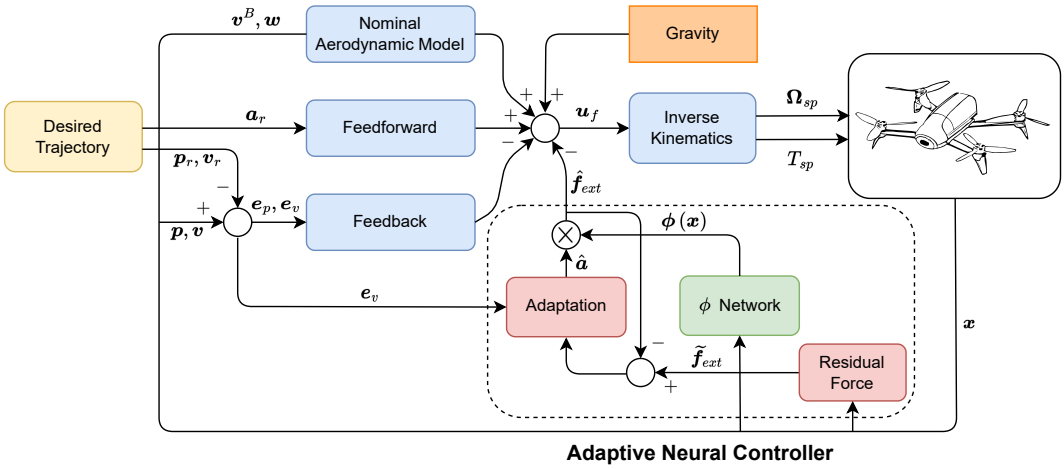



Figure 2: Control Diagram of the Adaptive Neural Controller

In this adversarial loss the discriminator network $\boldsymbol{h}$ is competing against $\boldsymbol{\phi}$ in a zero-sum game. The network $\boldsymbol{h}$ attempts to predict the condition $k$ from the output of the network $\boldsymbol{\phi}(\boldsymbol{x})$, while $\boldsymbol{\phi}$ attempts to approximate the residual force measurement $y$ while making the job of $h$ harder. The influence of the discriminator network $h$ is controlled through the regularization term $\alpha \geq 0$. $\mathcal{L}_{\boldsymbol{h}}$ is the loss function of $h$, typically a cross-entropy loss function is used to train classification models:

$$\mathcal{L}_{\boldsymbol{h}} = -\sum_{i \in B} \delta_{k=j} \odot \log\left(\boldsymbol{h}\left(\boldsymbol{\phi}\left(x_i^k\right)\right)\right) \tag{11}$$

The cross-entropy loss compares the log of the output of the discriminator network $\boldsymbol{h}\left(\boldsymbol{\phi}\left(x_i^k\right)\right)$, a vector containing the classification probabilities of each of the $K$ conditions, with the ground truth classification vector $\delta_i$. For example, if a data point is collected with no propeller damage with index $k = 1$ the corresponding ground truth vector is $\delta_i = [1, 0, 0]$.

The DAIML algorithm developed in [27] is shown in algorithm 1. DAIML consists of three main steps: (I) **the adaptation step** solves the least squares problem on the adaptation batch $B^a$ to find the optimal latent variable $\boldsymbol{A}^*$ with a fixed network $\boldsymbol{\phi}$, (II) **the $\boldsymbol{\phi}$ training step** updates the parameters of the network $\boldsymbol{\phi}$ using Stochastic Gradient Descent (SGD) based on the adversarial loss over the training batch $\boldsymbol{B}^t$ with $\boldsymbol{A}^*$, (III) **the $\boldsymbol{h}$ training step** updates the parameters of the discriminator network $h$ using SGD based on the cross entropy loss over the training batch $\boldsymbol{B}^t$.

Nevertheless, training adversarial networks is a challenging task due to problems such as mode collapse, vanishing gradients and unstable convergence. Fortunately, Generative Adversarial Networks (GANs) have been extensively researched and various well-documented features have been shown to improve training. These improvements have been implemented in DAIML they include (1) normalization of the latent variable $||\boldsymbol{A}^*|| > \gamma$ to improve robustness, (2) spectral normalization of the weights of the discriminator network $\boldsymbol{W}_h = \boldsymbol{W}_h/\sigma(\boldsymbol{W}_h)$ to improve the stability, (3) training the networks $\boldsymbol{\phi}$ and $\boldsymbol{h}$ in an alternating manner as well as training the discriminator less frequently to improve convergence.

**Algorithm 1:** DAIML

**Hyperparameters:** $\alpha \geq 0,\ \gamma > 0,\ 0 < \eta \leq 1$
**Data:** $\boldsymbol{D} = \{\boldsymbol{D}^1, \ldots, \boldsymbol{D}^K\}$
**Result:** trained neural network $\boldsymbol{\phi}$ and $\boldsymbol{h}$

**while** *not converged* **do**
  Randomly sample $\boldsymbol{D}^k \in \boldsymbol{D}$
  Randomly sample disjoint batches $\boldsymbol{B}^a, \boldsymbol{B}^t \in \boldsymbol{D}^k$
  Solve $\boldsymbol{A}^*(\phi) = \arg\min_{\boldsymbol{A}} \sum_{i \in B^a} \left\|y_i^k - \boldsymbol{\phi}\left(x_i^k\right)\boldsymbol{A}\right\|$
  Train $\boldsymbol{\phi}$ with loss: $\mathcal{L}_{\boldsymbol{\phi}} = \sum_{i \in B^t} \left(\left\|y_i^k - \boldsymbol{\phi}\left(x_i^k\right)\boldsymbol{A}^*\right\| - \alpha \cdot \mathcal{L}_{\boldsymbol{h}}\left(\boldsymbol{h}\left(\boldsymbol{\phi}\left(x_i^k\right)\right), k\right)\right)$
  **if** $X \sim U(0,1) \leq \eta$ **then**
    Train $\boldsymbol{h}$ with cross-entropy loss: $\mathcal{L}_{\boldsymbol{h}} = -\sum_{i \in B^t} \delta_i \odot \log\left(\boldsymbol{h}\left(\boldsymbol{\phi}\left(x_i^k\right)\right)\right)$
  **end**
**end**

### 3.2.2 Online Adaptation

An online adaptation law based on a Kalman-filter estimator is used to update the estimate of the linear coefficients $\hat{\boldsymbol{A}}$ through Equation 12. While in the offline training phase, the optimal coefficients $\boldsymbol{A}^*$ are obtained by minimizing the least squares force prediction error. In the online adaptation phase, the goal is to minimize the position tracking error. Thus, the linear coefficients are updated based on a composite er-

http://www.imavs.org/

http://www.imavs.org/

ror, consisting of the force prediction error and the tracking error, improving the trajectory tracking performance of the controller. Additionally, the Kalman-filter estimator automatically tunes the gain matrix $P$ given by Equation 13. This allows the online adaptation law to estimate the complex latent variable in the presence of varying uncertainties, alleviating the need for constant excitation for accurate estimation.

$$\dot{\hat{\boldsymbol{A}}} = \underbrace{-\lambda\hat{\boldsymbol{A}}}_{\text{regularization}} \underbrace{-\boldsymbol{P}\phi(\boldsymbol{x})^\top \boldsymbol{R}^{-1}(\phi(\boldsymbol{x})\hat{\boldsymbol{A}} - \boldsymbol{y})}_{\text{prediction error}} + \underbrace{\boldsymbol{P}\phi(\boldsymbol{x})^\top \boldsymbol{s}}_{\text{tracking error}} \tag{12}$$

$$\dot{\boldsymbol{P}} = -2\lambda\boldsymbol{P} + \boldsymbol{Q} - \boldsymbol{P}\phi(\boldsymbol{x})^\top \boldsymbol{R}^{-1} + \phi(\boldsymbol{x})\boldsymbol{P} \tag{13}$$

In practice, the adaptation law is implemented in a digital system which requires the discrete version of the Kalman filter. This discrete Kalman filter suffers from numerical stability and convergence issues, particularly in the covariance matrix $P$. As discussed in [27], coarse integration step sizes of the continuous time adaptation law would sometimes result in $P$ becoming non-positive-definite. Their proposed solution, splitting the adaptation law into two steps: a propagation step and an update step is implemented.

### 3.3 *Trajectory Generation*

The desired trajectory of the quadrotor is defined using a spline that minimizes snap, the 4th derivative of position with respect to time $\ddddot{\boldsymbol{p}}$. Minimum snap trajectories are desirable for quadrotor tracking since motor commands and attitude accelerations are proportional to snap. Moreover, polynomial splines are easily differentiable and useful when calculating the velocity and acceleration of the reference trajectory.

Given a list of $N$ waypoints $\boldsymbol{W} = [\boldsymbol{W}_1, \ldots, \boldsymbol{W}_N]$ and $W_n \in \mathbb{R}^3$ the desired position of the quadrotor is given by $\boldsymbol{p}_m = [p_{m_x}(t), p_{m_y}(t), p_{m_z}(t)]$ with each axis described by one spline consisting of $M = N - 1$ polynomials. The minimum snap spline of each axis is obtained by solving the minimization problem:

$$\min_{p_1,\cdots,p_M} \sum_{m=1}^{M} \int_0^T \| \ddddot{p}_m(t) \| \, dt \tag{14}$$

where $T$ is the amount of time for each waypoint segment. Given that the distance between waypoints is arbitrary using the same amount of time for all segments severely constrains the solution quality. To improve the overall solution the segment times are included in the cost function in Equation 15 and allowed to vary. However, changing the segment times also changes the cost function resulting in a non-convex optimization problem, an initial guess of segment times is required and then iteratively refined using gradient descent to reach a minimum.

$$\min_{p_1,T_1,\cdots,p_M,T_M} \sum_{m=1}^{M} \int_0^{T_m} \| \ddddot{p}_m(t) \| \, dt + \mu T_m \tag{15}$$

Subject to:

$$\boldsymbol{p}_m(0) = \boldsymbol{W}_m, \quad \boldsymbol{p}_m(T_m) = \boldsymbol{W}_{m+1} \tag{16}$$

$$\dot{\boldsymbol{p}}_1(0) = \boldsymbol{0}, \quad \dot{\boldsymbol{p}}_M(T_M) = \boldsymbol{0} \tag{17}$$

$$\dot{\boldsymbol{p}}_m(T_m) = \dot{\boldsymbol{p}}_{m+1}(0), \quad \ddot{\boldsymbol{p}}_m(T_m) = \ddot{\boldsymbol{p}}_{m+1}(0) \tag{18}$$

$$T_m \geq 0, \tag{19}$$

where $\mu > 0$ is time penalty factor. The time penalty factor influences the trade-off between minimizing snap and total trajectory time.

## 4 EXPERIMENTAL SETUP

The quadrotor platform used is the Parrot Bebop 1, and is equipped with a Parrot P7 dual-core Cortex A9 CPU and an MPU6050 IMU. Relevant parameters of the Parrot Bebop 1 are included in Table 1.

The Parrot Bebop 1 has a thrust-to-weight ratio of $T/W \approx 1.7$ significantly lower than that of modern custom drones, limiting the maximum speeds that can be reached. Despite this drawback the Parrot Bebop 1 has certain advantages for our application: (I) the aging onboard CPU has limited computational capacity forcing the online implementation of the adaptive neural controller to be computationally efficient (II) the relatively large body fairing introduces complex aerodynamic disturbances even at low speeds which are hard to model (III) it is equipped out-of-the-box with rpm sensors on all propellers.

Table 1: Parrot Bebop 1 Parameters

| Parameter | Unit | Value |
|---|---|---|
| $m$ | $[kg]$ | $0.390$ |
| $(l_x, l_y)$ | $[m]$ | $(9.5e^{-2}, 7.75e^{-2})$ |
| $\boldsymbol{I}$ | $[gm^2]$ | $diag\left(9.06e^{-4}, 1.24e^{-3}, 2.05e^{-3}\right)$ |
| $c_t$ | $[N^2]$ | $4.36e^{-8}$ |
| $c_\tau$ | $[Nm^2]$ | $6.60e^{-12}$ |
| $(k_x, k_y, k_z)$ | $[kg/s]$ | $\left(1.08e^{-5}, 9.65e^{-6}, 2.79e^{-5}\right)$ |
| $k_h$ | $[kg/m]$ | $6.26e^{-2}$ |

The onboard software of the Parrot Bebop 1 has been flashed with the Paparazzi-UAV open source autopilot project[1] written in $C$. Designed with autonomous applications in mind and highly customizable, it allows for the straightforward implementation of modules onboard. The offline DNN training is performed in Python using Pytorch[2] an open-source deep learning framework in combination with WandB[3] a powerful tool for managing and tracking machine learning experiments.

[1] https://wiki.paparazziuav.org/wiki/Main_Page
[2] https://pytorch.org/
[3] https://wandb.ai/site

The test flights are performed in the CyberZoo, an indoor $10 \times 10 \times 7\ m^3$ square volume equipped with an OptiTrack motion capture system. The OptiTrack measurements are fused with the IMU measurements onboard using an Extended Kalman filter (EKF) for accurate state estimation. A diagram of the experimental setup is shown in Figure 3.

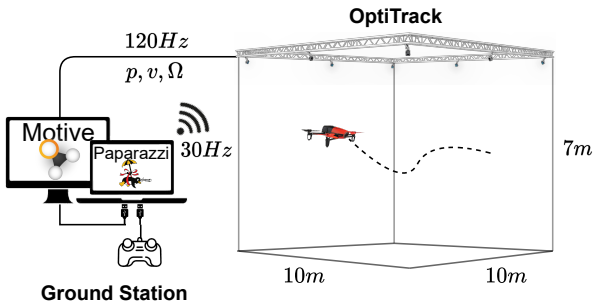



Figure 3: Experimental Setup

## 5 RESULTS

### 5.1 Dataset

#### 5.1.1 Data Collection

The DAIML algorithm requires data to train the representation $\phi$. Data collection consists of logging relevant time-stamped drone states as the quadrotor tracks minimum snap trajectories using a conventional velocity PID controller. These trajectories are generated from 10 randomly sampled waypoints using the method outlined in subsection 3.3. If at any point throughout the generated trajectory, the position and a tracking margin exceed the limits of the CyberZoo a different set of waypoints is sampled till a suitable trajectory is found. To gather data across a wide range of velocities several minimum snap trajectories are generated with varying time factors. However, all trajectories will travel roughly the same distance but with varying times, resulting in slower trajectories having more data points than faster trajectories. To ensure a more uniform distribution of data points across velocities multiple trajectories are logged at higher speeds, preventing the network from learning a bias towards a certain velocity regime due to it being over-represented in the data collected.

To simulate propeller damage all 4 propellers of the bebop are cut at the tip by approximately $2mm$ and $4mm$ corresponding to a $5.3\%$ and $10.5\%$ reduction in blade radius. Replacing the smooth elliptical propeller tips with rough rectangular tips increases the wing tip vortices generated by the fast-spinning propellers. This results in a reduction in propeller lift and an increase in propeller drag, altering the aerodynamic effects acting on the quadrotor. Data is collected on the training and validation trajectories outlined in Table 2 for all 3 propeller conditions: (1) no damage, (2) slight damage ($2mm$) and (3) significant damage ($4mm$).

Table 2: Training and Validation Dataset

| | Time Factor ($\mu$) | # | Total Time ($T_{tot}$) [s] | Datapoints |
|---|---|---|---|---|
| **Train** | 0.2 | 1 | 270 | 7547 |
| | 0.4 | 2 | 263 | 7486 |
| | 0.6 | 3 | 311 | 8883 |
| | 0.8 | 4 | 282 | 7968 |
| | 1.0 | 5 | 287 | 7923 |
| | 1.2 | 4 | 205 | 5691 |
| **Val** | 0.3 | 1 | 182 | 4815 |
| | 0.7 | 1 | 81 | 2120 |
| | 1.1 | 1 | 49 | 1223 |
| **Total** | - | 22 | $1,930$ | $61,624$ |

#### 5.1.2 Data Processing

The unmodeled forces acting on the quadrotor are obtained by rearranging Equation 1 to solve for $f_{res}$. The quadrotor acceleration is required but not logged, instead, it is computed using 1st-order central difference of the velocity measurements. However, the velocity measurements contain significant noise which is further amplified when calculating the acceleration derivative. A zero-phase "filtfilt" filter is used to smooth out the velocity measurements to reduce the noise in the unmodeled force without introducing lag.

The dataset is divided into the 3 different propeller conditions $\boldsymbol{D} = \{\boldsymbol{D}^1, \boldsymbol{D}^2, \boldsymbol{D}^3\}$ with each dataset containing $N_k$ input-output pairs $\boldsymbol{D}^k = \{\boldsymbol{x}_i^k, \boldsymbol{y}_i^k\}_{i=1}^{N_k}$ of the relevant drone states and a noisy measurement of the unmodeled forces $\boldsymbol{y} = \boldsymbol{f}_{res} + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ encompasses all source of noise.

### 5.2 Training

The DAIML algorithm outlined in subsubsection 3.2.1 is used to train an effective representation $\boldsymbol{\phi}$ of the propeller damage invariant aerodynamic effects from the data collected. The inputs of the network $\boldsymbol{\phi}$ are the velocity, quaternion and rpm states $\boldsymbol{x} = [\boldsymbol{v}, \boldsymbol{q}, \boldsymbol{rpm}]$, while the output representation is concatenated $\boldsymbol{y} = [y_1, y_2, y_3, 1]$ to provide a constant term for the adaptation. The architecture of the network $\boldsymbol{\phi}$ consists of $[11, 50, 60, 50, 4]$ fully connected layers with Rectified Linear Unit (ReLU) activation functions as shown in Figure 4. Similarly, the discriminator network $h$ consists of $[4, 128, 3]$ fully connected layers with ReLU activation functions.

The hyperparameters used during training are reported in Table 3, obtained from the sensitivity analysis. The commonly used Adam optimizer, an extended version of stochastic gradient descent, is used to update the weights of both networks $\boldsymbol{\phi}$ and $\boldsymbol{h}$ with learning rates $lr_\phi = 2e^{-3}$ and $lr_h = 4e^{-4}$ and batch sizes $B^t = 128$ and $B^a = 512$ respectively. The normalization term $\gamma = 17$ ensures that the learned adaptation coefficients remain bounded, important for fast adaptation in the online phase. The regularization term $\alpha$

http://www.imavs.org/

http://www.imavs.org/

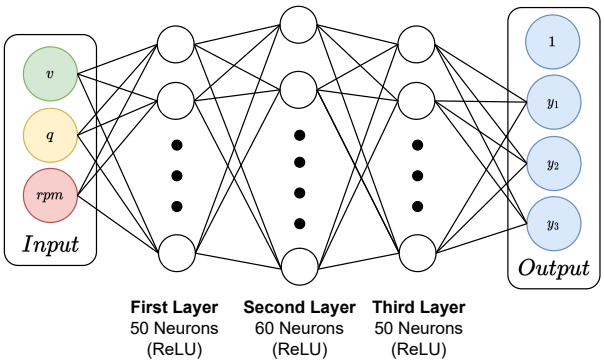



Figure 4: Neural Network $\phi$ Architecture

influences the degree of adversarial training. A regularization value of $\alpha = 0.1$ was found to balance the $\phi$ network performance and disentangle the learned linear coefficients into separate propeller conditions. The stability of the discriminator network $\boldsymbol{h}$ is improved by using a discriminator training frequency of $\eta = 0.75$, combined with spectral normalization of the discriminator weights.

Table 3: Hyperparameters of DAIML Algorithm

| Hyperparameter | Symbol | Value |
|---|---|---|
| **$\phi$ Network** | | |
| Learning Rate | $lr_\phi$ | $2e^{-3}$ |
| Normalization | $\gamma$ | 17 |
| Training Batch Size | $B^t$ | 128 |
| **$\boldsymbol{h}$ Network** | | |
| Learning Rate | $lr_h$ | $4e^{-4}$ |
| Regularization | $\alpha$ | 0.1 |
| Training Frequency | $\eta$ | 0.75 |
| Adaptation Batch Size | $B^a$ | 512 |

The high dimensional adaptation coefficients are projected into a 2-dimensional plane using the t-Distributed Stochastic Neighbor Embedding (t-SNE) dimensionality reduction technique. The t-SNE algorithm preserves local similarities in the higher dimensional data enabling visualization of clusters and patterns. The t-SNE plots of the training adaptation coefficients $\boldsymbol{A}^*$ with $\alpha = 0.1$ and $\alpha = 0$ are shown in Figure 7. With $\alpha = 0$ corresponding to the non-adversarial training case. Adversarial training enforces the adaptation coefficients into explaining clusters that contain the propeller damage condition information, ensuring the trained representation $\phi$ is condition invariant.

#### 5.2.1 Unmodeled Force Prediction

The unmodeled force predictions of the network $\phi$ are shown on the training, testing and validation data for the no propeller damage condition. The adaptation coefficients learned during training corresponding to the no propeller damage condition are used to compute the unmodeled force predictions $\hat{\boldsymbol{f}}_{ext} = \phi(\boldsymbol{x})\,\boldsymbol{A}^*$.

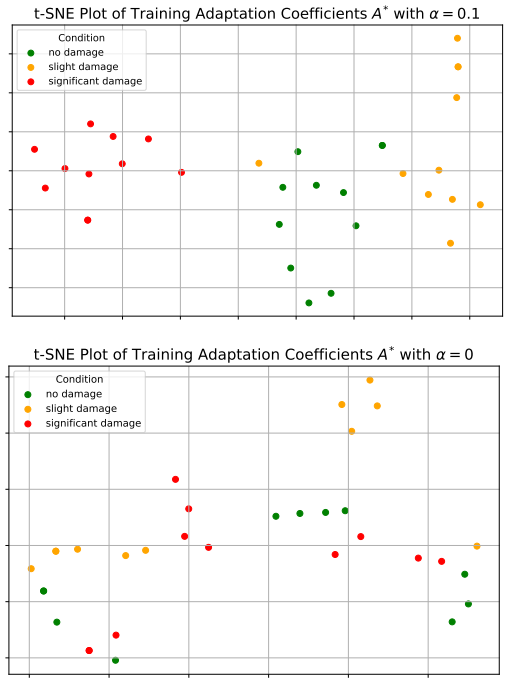



Figure 7: t-SNE plots of the Training Adaptation Coefficients $\boldsymbol{A}^*$ with $\alpha = 0.1$ and $\alpha = 0$

The $\phi$ network successfully learns a representation that captures the main features of the unmodeled forces acting on the quadrotor, providing a filtering zero-lag estimate of these unmodeled forces. Even at the relatively slow maximum speeds the Parrot Bebop 1 can achieve the conventional aerodynamic model fails to model aerodynamic forces in the order of $f_{res} \approx 0.3$ in the x-y direction and $f_{res} \approx 0.7$ in the z-direction, which the network $\phi$ can model.

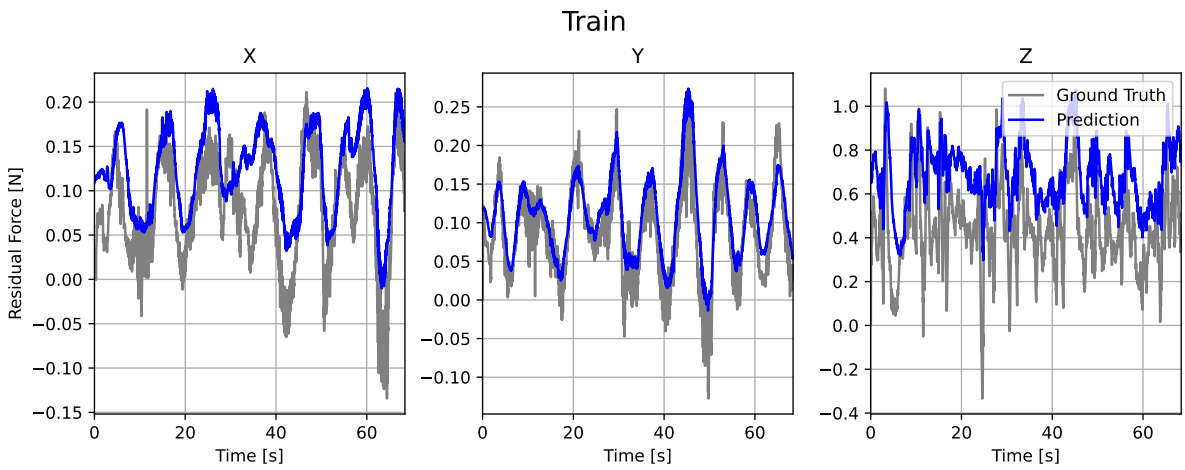
While the network captures the main features of the unmodeled forces they may be off by a certain offset, this is especially noticeable on the validation data. This may be caused by either the different controller architecture between train/test and validation or because a different quadrotor of the same model is used. Substantiating the need to adapt coefficients online to account for differences between training data and real-life deployment.

### 5.3 *Tracking Performance*

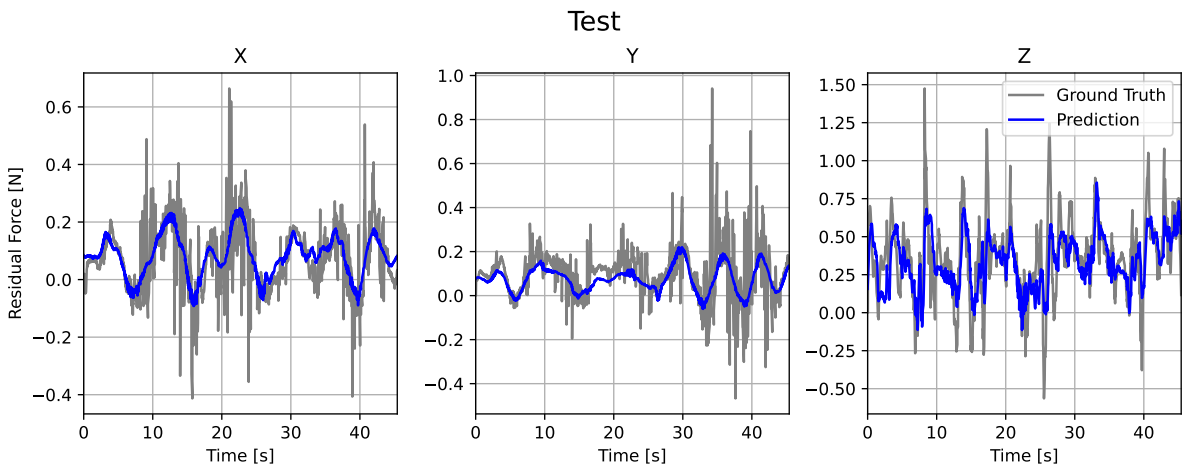
The tracking performance of the neural adaptive controller is compared to that of the nonlinear controller in Equation 8 without the adaptation term and the baseline velocity PID controller. The performance of these controllers is measured by tracking a randomized minimum snap trajectory with time factor $\mu = 0.7$ and $\mu = 1.1$, shown in Figure 9.

The adaptive neural controller outperforms the other controllers by a substantial margin, achieving a tracking error one order of magnitude lower than the conventional velocity PID controller and approximately $\approx 60\%$ lower than the nonlin-

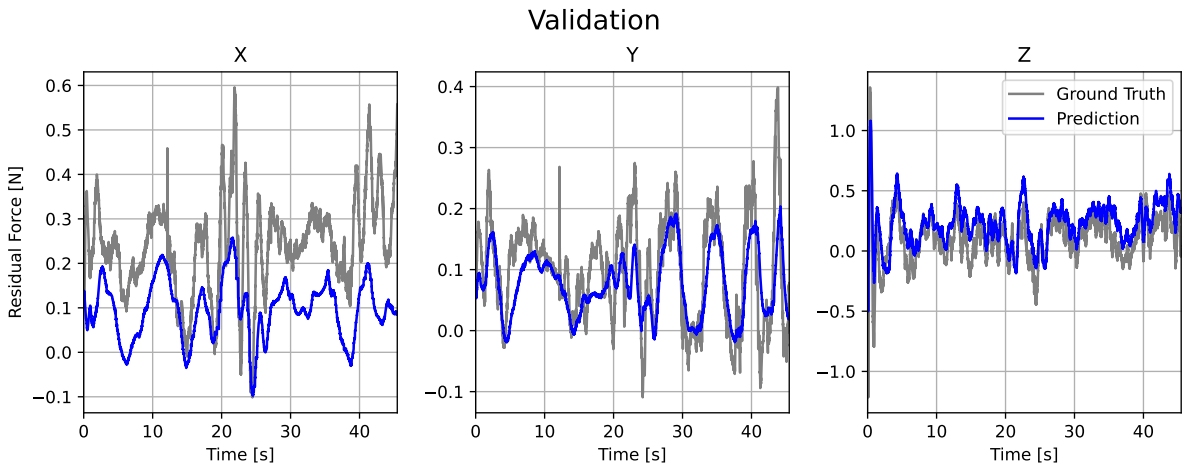
http://www.imavs.org/




(a) Training Data




(b) Testing Data




(c) Validation Data

Figure 8: Unmodeled Force Predictions $\hat{\boldsymbol{f}}_{ext} = \boldsymbol{\phi}(\boldsymbol{x})\,\boldsymbol{A}^*$ for No Propeller Damage Condition

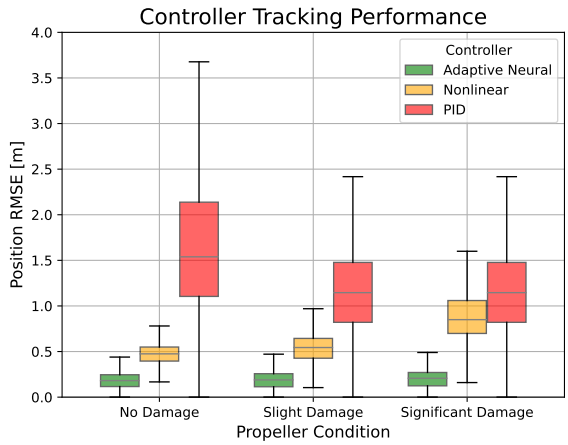



Figure 9: Controller Tracking Performance Comparison on Minimum Snap Trajectory with $\mu = 0.7$

ear controller. Most importantly, the tracking performance of the adaptive neural controller barely degrades with increasing propeller damage in contrast to the other controllers, with a less than $0.03$ $[m]$ increase in tracking error. This showcases the ability of the adaptive neural controller to adapt to the current propeller condition flown.

The trajectories flown using the neural adaptive controller and the nonlinear controller are shown in Figure 10 for all propeller damage conditions.

## 6 DISCUSSION & RECOMMENDATIONS

The adaptive neural controller achieves impressive tracking results with minimal susceptibility toward propeller damage. The neural network $\phi$ successfully learns a representation of the unmodeled aerodynamic forces acting on the quadrotor.

### 6.1 *Computational Efficiency*

Most quadrotor adaptive controllers in literature require powerful embedded computers or even a ground station. The adaptive neural controller runs onboard an aging dual-core CPU without breaking a sweat, a testament to its computational efficiency. The efficient $C$ implementation of the network inference and adaptation allows the controller to run onboard at $\approx 30$ $[Hz]$ with a CPU load of $\approx 60\%$ but could even run at a much faster $\approx 100$ $[Hz]$ at a CPU load of $\approx 80\%$. For the offline learning phase, the DAIML algorithm is equally fast at learning an efficient representation of the aerodynamic forces taking approximately $\approx 4m$ to train a network from scratch on a laptop.

### 6.2 *Influence of Propeller Damage on Aerodynamic Model*

The effect of propeller damage on the residual forces is mainly limited to the vertical direction, with little change in the lateral residual forces. This causes some instability in the adversarial training procedure albeit these problems could be solved through careful hyperparameter selection. Increasing speed and propeller rpms or increasing propeller damage should make the influence of propeller condition more noticeable, however, we are ultimately limited by the thrust-to-weight ratio of the quadrotor and the available space of the CyberZoo.

### 6.3 *Noisy Measurements*

The $\phi$ network is susceptible like any other neural network to the quality of the data it trains on. To compute the unmodeled force measurements acting on the drone the velocity measurements are differentiated, which further amplifies the noise. The OptiTrack velocity measurements experienced high noise, frequency dropouts and occasional asynchronous timing. Despite efforts to filter and clean the training and testing data some of noise and errors made it through into the ground truth data, degrading the quality of the learned representation $\phi$.

The poor quality of the OptiTrack measurements also influences the online adaptation. Significant filtering of the velocity measurements is required before applying backward finite difference to obtain usable measurements of the residual force $\boldsymbol{y}$ which have significant lag. This may cause oscillatory behaviour as can be seen in some of the trajectories in Figure 10.

### 6.4 *Recommendations*

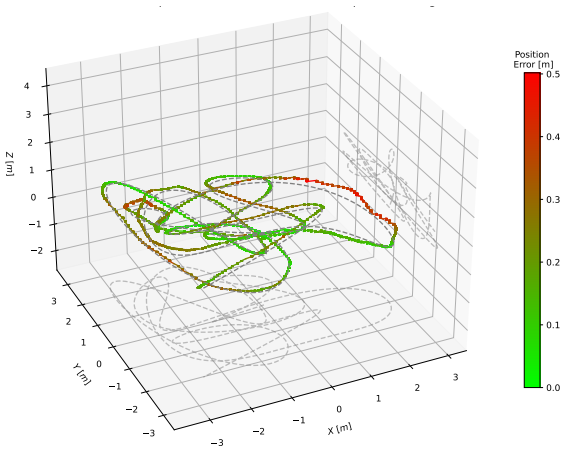
Future work should focus on addressing the challenge of significant noise in the residual force measurements in the online adaptation. The quality of the residual force measurements could be improved by implementing a properly tuned Kalman-filter with accelerations as states. Additionally, accelerometer measurement noise can be mitigated by using a modern IMU or in the case of the OptiTrack velocity measurements using ball infrared markers and limiting reference trajectories to remain in areas with good OptiTrack coverage.

In terms of improving the architecture of the adaptive neural controller, research in artificial intelligence has shifted towards using transformers which outperform conventional fully connected or convolutional network architectures. Transformers process a sequence of inputs in contrast to a single input. Furthermore, transformers have recently been employed in Generative Adversarial Networks in [29] similar to DAIML achieving state-of-the-art performance.

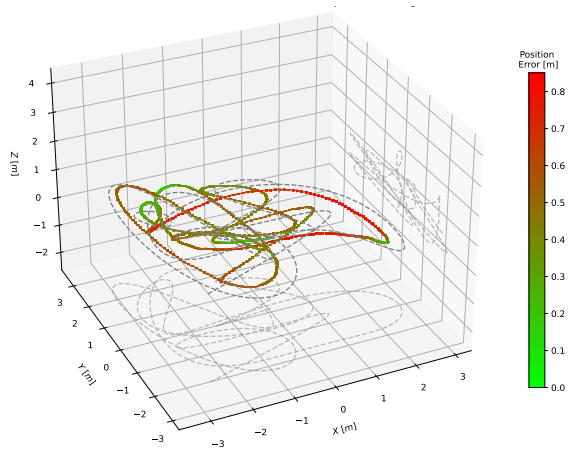
The effect of propeller damage on the residual forces was found to not be as significant as expected, diminishing the benefits of using an adaptive controller. Instead adapting to flying the Parrot Bebop 1 with and without bumpers attached would be a better application. We believe that this adaptive neural controller would be best suited for morphing or VTOL UAVs which have multiple distinct aerodynamic models which are challenging to model during the transition phase.

http://www.imavs.org/

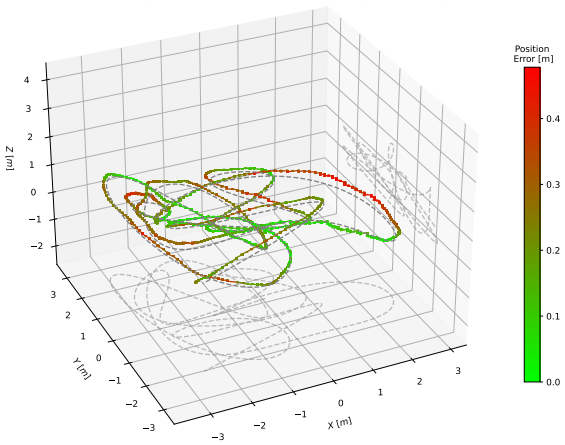
http://www.imavs.org/



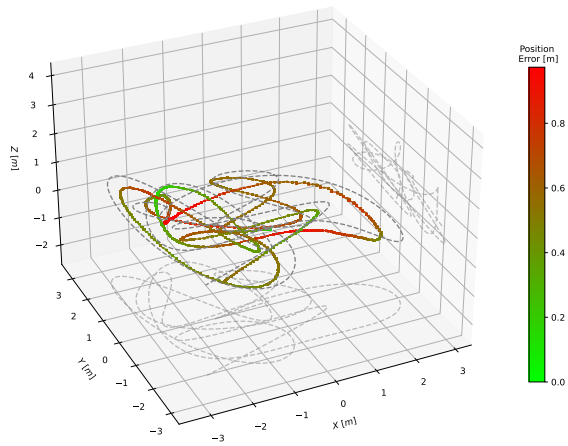

(a) Adaptive Neural Controller w/ No Propeller Damage



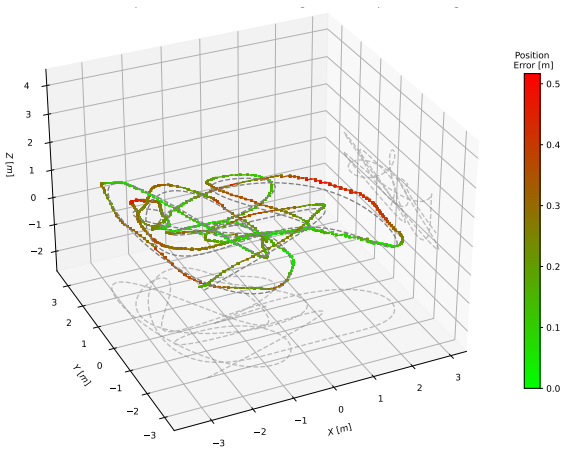

(b) Nonlinear Controller w/ No Propeller Damage



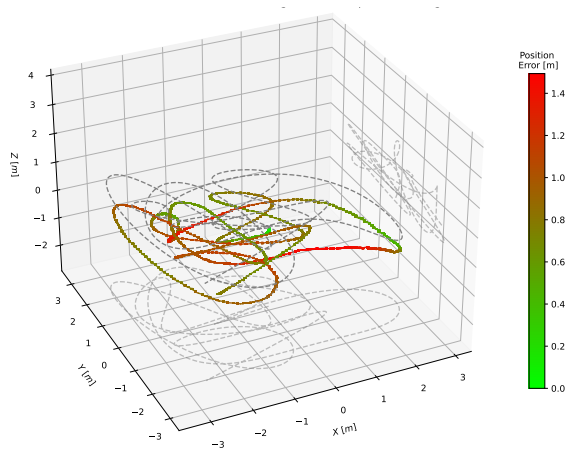

(c) Adaptive Neural Controller w/ Slight Propeller Damage




(d) Nonlinear Controller w/ Slight Propeller Damage




(e) Adaptive Neural Controller w/ Significant Propeller Damage




(f) Nonlinear Controller w/ Significant Propeller Damage

Figure 10: Comparison of Trajectories flown using Adaptive Neural Controller and Nonlinear Controller for all Propeller Conditions

http://www.imavs.org/

## 7 CONCLUSION

The "Neural-Fly" controller from [27] is a lightweight yet powerful adaptive neural controller that combines pre-trained condition invariant representations of the unmodeled forces using Domain Adversarially Invariant Meta-Learning (DAIML) with fast online adaptation. While it was originally applied to wind speed conditions, we apply this adaptive neural architecture to various degrees of propeller damage a common condition in real-work flight, demonstrating the versatility of this controller architecture in adapting to various conditions. We show that the adaptive neural architecture is a computationally efficient and accurate trajectory-tracking controller architecture. The adaptive neural controller is deployed fully onboard the Parrot Bebop 1 without the need to attach powerful embedded computers, showcasing the computational efficiency of our controller implementation. We perform a sensitivity analysis on the hyperparameters of the DAIML algorithm consisting of 30 runs to determine the optimal hyperparameters using Bayesian search, further improving the performance of the controller. We show that the learned propeller damage invariant representation $\phi$ using DAIML successfully predicts a filtered zero-lag prediction of the unmodeled forces even at the limited speeds and propeller rpms of the experimental setup. The adaptive neural controller successfully adapts to the various degrees of propeller damage with minimal degradation in tracking performance significantly outperforming the nonlinear and velocity PID baseline controllers.

## REFERENCES

[1] Theodore Tzanetos, MiMi Aung, J. Balaram, Havard Fjrer Grip, Jaakko T. Karras, Timothy K. Canham, Gerik Kubiak, Joshua Anderson, Gene Merewether, Michael Starch, Mike Pauken, Stefano Cappucci, Matthew Chase, Matthew Golombek, Olivier Toupet, Marshall C. Smart, Stephen Dawson, Erick Blandon Ramirez, Johnny Lam, Ryan Stern, Nacer Chahat, Joshua Ravich, Robert Hogg, Benjamin Pipenberg, Matthew Keennon, and Kenneth H. Williford. Ingenuity mars helicopter: From technology demonstration to extraterrestrial scout. In *2022 IEEE Aerospace Conference (AERO)*. IEEE, March 2022.

[2] Dara Kerr and Richard Nieva. Drones, sun — and a strong will — elevate Rwanda's health care.

[3] Guillem Torrente, Elia Kaufmann, Philipp Foehn, and Davide Scaramuzza. Data-Driven MPC for Quadrotors. In *IEEE Robotics and Automation Letters*, volume 6 of *2*, pages 3769–3776. IEEE, March 2021.

[4] James Svacha, Kartik Mohta, and Vijay Kumar. Improving quadrotor trajectory tracking by compensating for aerodynamic effects. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 860–866, June 2017.

[5] Patricia Ventura Diaz and Steven Yoon. High-Fidelity Computational Aerodynamics of Multi-Rotor Unmanned Aerial Vehicles. In *2018 AIAA Aerospace Sciences Meeting*, Kissimmee, Florida, January 2018. American Institute of Aeronautics and Astronautics.

[6] Ivan Lopez-Sanchez, Francisco Rossomando, Ricardo Pérez-Alcocer, Carlos Soria, Ricardo Carelli, and Javier Moreno-Valenzuela. Adaptive trajectory tracking control for quadrotors with disturbances by using generalized regression neural networks. *Neurocomputing*, 460:243–255, October 2021.

[7] Mahdis Bisheban and Taeyoung Lee. Geometric Adaptive Control with Neural Networks for a Quadrotor UAV in Wind fields, March 2019.

[8] Qiyang Li, Jingxing Qian, Zining Zhu, Xuchan Bao, Mohamed K. Helwa, and Angela P. Schoellig. Deep Neural Networks for Improved, Impromptu Trajectory Tracking of Quadrotors, October 2016.

[9] Aminurrashid Noordin, Ariffanan Basri, and Zaharuddin Mohamed. Simulation and experimental study on PID control of a quadrotor MAV with perturbation. *Bulletin of Electrical Engineering and Informatics*, 9, October 2020.

[10] Luís Martins, Carlos Cardeira, and Paulo Oliveira. Linear Quadratic Regulator for Trajectory Tracking of a Quadrotor. *IFAC-PapersOnLine*, 52(12):176–181, January 2019.

[11] J.J.E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, 1991.

[12] Daewon Lee, H. Jin Kim, and Shankar Sastry. Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. *International Journal of Control, Automation and Systems*, 7(3):419–428, June 2009.

[13] P.V. Kokotovic. The joy of feedback: nonlinear and adaptive. *IEEE Control Systems Magazine*, 12(3):7–17, June 1992. Conference Name: IEEE Control Systems Magazine.

[14] S. Sieberling, Q. P. Chu, and J. A. Mulder. Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction. *Journal of Guidance, Control, and Dynamics*, 33(6):1732–1742, November 2010. Publisher: American Institute of Aeronautics and Astronautics.

[15] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011*

http://www.imavs.org/

*IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011.

[16] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories. In *Robotics and Automation Letters (RA-L)*, volume 3 of *2*, pages 620–626. IEEE, April 2018.

[17] Ezra Tal and Sertac Karaman. Accurate Tracking of Aggressive Quadrotor Trajectories using Incremental Nonlinear Dynamic Inversion and Differential Flatness. In *IEEE Transactions on Control Systems Technology*, volume 29 of *3*, pages 1203–1218. IEEE, June 2020. arXiv:1809.04048 [cs] type: article.

[18] Sihao Sun, Angel Romero, Philipp Foehn, Elia Kaufmann, and Davide Scaramuzza. A Comparative Study of Nonlinear MPC and Differential-Flatness-Based Control for Quadrotor Agile Flight. In *IEEE Transactions on Robotic*, pages 3357–3373. IEEE, December 2022. arXiv:2109.01365 [cs] type: article.

[19] Prasanth Kotaru, Ryan Edmonson, and Koushil Sreenath. Geometric L1 Adaptive Attitude Control for a Quadrotor Unmanned Aerial Vehicle, March 2020.

[20] Drew Hanover, Philipp Foehn, Sihao Sun, Elia Kaufmann, and Davide Scaramuzza. Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors. *IEEE Robotics and Automation Letters*, 7(2):690–697, April 2022.

[21] Zachary T. Dydek, Anuradha M. Annaswamy, and Eugene Lavretsky. Adaptive Control of Quadrotor UAVs: A Design Trade Study With Flight Evaluations. *IEEE Transactions on Control Systems Technology*, 21(4):1400–1406, July 2013. Conference Name: IEEE Transactions on Control Systems Technology.

[22] Xichen Shi, Patrick Spieler, Ellande Tang, Elena-Sorina Lupu, Phillip Tokumaru, and Soon-Jo Chung. Adaptive Nonlinear Control of Fixed-Wing VTOL with Airflow Vector Sensing. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5321–5327, May 2020. ISSN: 2577-087X.

[23] Alexander Spitzer and Nathan Michael. Feedback Linearization for Quadrotors with a Learned Acceleration Error Model. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6042–6048, May 2021.

[24] Fu-Chuang Chen and H.K. Khalil. Adaptive control of a class of nonlinear discrete-time systems using neural networks. *IEEE Transactions on Automatic Control*, 40(5):791–801, May 1995.

[25] A. Kondratiev and Y. Tiumentsev. Inverse Dynamics Approach to Adaptive Damage-Tolerant Control for Unmanned Aerial Vehicles. In *Proceedings of the International Micro Air Vehicle conference and flight competition 2011 summer edition*, 2011.

[26] Suneel Belkhale, Rachel Li, Gregory Kahn, Rowan McAllister, Roberto Calandra, and Sergey Levine. Model-Based Meta-Reinforcement Learning for Flight with Suspended Payloads. *IEEE Robotics and Automation Letters*, 6(2):1471–1478, April 2021.

[27] Michael O'Connell, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural-Fly Enables Rapid Learning for Agile Flight in Strong Winds. In *Science Robotics*, volume 7 of *66*, page eabm6597. American Association for the Advancement of Science, May 2022.

[28] Robin Ferede, Guido C. H. E. de Croon, Christophe De Wagter, and Dario Izzo. An adaptive control strategy for neural network based optimal quadcopter controllers, 2023.

[29] Drew A. Hudson and C. Lawrence Zitnick. Generative Adversarial Transformers, March 2022.