Adaptive Neural Control

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

July 2023

Mauro Villanueva Aguado

Delft University of Technology

TU Delft

Adaptive Neural Control

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

by

Mauro Villanueva Aguado

in partial fulfillment of the requirements for the degree of
Master of Science in Aerospace Engineering
at the Delft University of Technology

to be defended publicly on Monday July 10th, 2023 at 10:00

Student Number:	4557824	
Project Duration:	December 2021 - July 2023	
Thesis Committee:	Dr. Ir. G.C.H.E. de Croon,	Chair, TU Delft
	Dr. Ir. C. de Wagter,	Supervisor, TU Delft
	Ir. R. Ferede,	Additional, TU Delft
	Dr. Ir. E.J.O. Schrama	Examiner, TU Delft

Front Page Image Courtesy of DRL © [1]

[1] https://thedroneracingleague.com/ [cited 10/12/2021]

TU Delft Delft University of Technology

Contents

Part I
Scientific Paper

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

Mauro Villanueva Aguado* and Christophe de Wagter†
Delft University of Technology, 2629 HS Delft, The Netherlands

Abstract

Executing quadrotor trajectories accurately and therefore safely is a challenging task. 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. Requiring additional embedded computers onboard, adding weight and requiring power. Given the limited computational resources onboard, a trade-off between accuracy and complexity must be considered. To this end, we implement "Neural-Fly" a lightweight adaptive neural controller to adapt to propeller damage, a common occurrence in real-world flight. The adaptive neural architecture consists of two components: (I) offline learning of a condition invariant representation of the aerodynamic forces through Deep Neural Networks (II) fast online adaptation to the current propeller condition 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 of revolutionizing a wide range of industries including cinematography, defence, agriculture, logistics. Crucially, UAVs require a trained operator to fly increasing operational costs, hindering the wider commoditization of UAVs. Autonomous UAVs make away the need of a trained operator, they 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], showcasing their incredible potential. However, the limited sensors and computational capabilities available onboard present significant challenges that must be solved to enable the widespread adoption of autonomous UAVs.

One of these challenges is executing trajectories accurately and thus safely in a wide range of conditions, essential for autonomous UAV applications in complex cluttered environments such as: search & rescue, aerial delivery/transport, drone racing or even space exploration. Accurate trajectory tracking of quadrotors, a common type of UAV with four rotors, is an specially challenging problem due to the complex aerodynamic disturbances at high speeds which consequently introduce large tracking errors. These aerodynamic effects are difficult to model, "they consist of the propeller lift and drag which are dependant on the induced airstream 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 significantly changing the aerodynamic model of the quadrotor. Designing such a controller does not only involve accurately tracking a trajectory across a range of propeller damage conditions but also designing one that remains computationally tractable to run onboard. Thus, a trade-off between model accuracy & complexity must be considered. To the best of our knowledge no prior work has been performed on the field of adaptive trajectory control to propeller damage, despite the fact that propeller damage either caused by collisions or degradation is a common occurrence in real-world situations.

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 their accuracy quickly degrades at higher speeds. Additionally, they 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 of solving in large compute clusters and ultimately are condensed into simplified models to be computationally tractable onboard. Advances in Neural Networks (NN) and small powerfull GPUs have enabled promising results 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 while requiring a fraction of the time and computation of CFD derived models. However implementing NN in a drone controller requires

*Correspondence: contact@maurovillanueva.eu, Master Candidate Control & Simulation Department, Faculty of Aerospace Engineering, TU Delft
†C.deWagter@tudelft.nl, Associate Professor at the Micro Air Vehicle Lab, Faculty of Aerospace Engineering, TU Delft

addressing the unpredictable nature of their output, remaining a largely unexplored challenge.

The remainder of 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. Followed by a discussion of these results and 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 unstable nonlinear 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 system, thus 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 good enough performance.

As quadrotors have become more powerfull and lightweight, capable of executing aggressive trajectories, nonlinear flight controllers have been proposed to cope with the nonlinearities 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 modelling 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 is Incremental Nonlinear Dynamic Inversion (INDI) [14], 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 impressive tracking performances 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 is thus susceptible to modeling mismatch, for example 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] a more than tenfold increase.

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 minimizing for tracking error. Similarly, the performance of NMPCs is further improved by adding an outer loop INDI controller and aerodynamic model. NMPCs are able to 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, specially 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]. Albeit, 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 powerfull 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 behaviour. 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 poor tracking performance [21].

On the other hand, 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, they should reflect important features of the underlying dynamics. Physics-based modelling 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 lack of excitation. Alternatively, random Fourier features can be used as basis functions as they can capture all underlying dynamics given a sufficient amount of features. However, the high-dimensional feature space may not be an efficient representation as features are redundant or irrelevant. 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 $\mathcal{L}_1$ 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 used NNs in MRACs to solve the identification problem of nonlinear systems creating a reference model [24, 25] whose weights are updated online based on the error between the system and model output. These networks where shallow and lacked pretraining, limiting their performance and requiring the initial guess of network weights to be close to the correct guess to ensure stability. More recently, a quadrotor has performed an impressive flip in wind using a 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 3-layer [6,3,3] architecture lacking pretraining.

Lately, meta-learning has drawn attention as 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 captures 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 lacking stability guarantees. 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 "NeuralFly" controller demonstrates state-of-the-art tracking performance in strong & varying wind conditions achieving a position RMSE of $9.4[cm]$ at wind speeds of 12.1 $[m/s]$ with little computational cost.

3 Methodology

In this paper vectors are denoted in lowercase bold letters $\mathbf{p}$ and matrices in uppercase bold letters $\mathbf{I}$, 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 $\boldsymbol{R} = [\boldsymbol{x}_B, \boldsymbol{y}_B, \boldsymbol{z}_B] \in \mathrm{SO}(3)$.

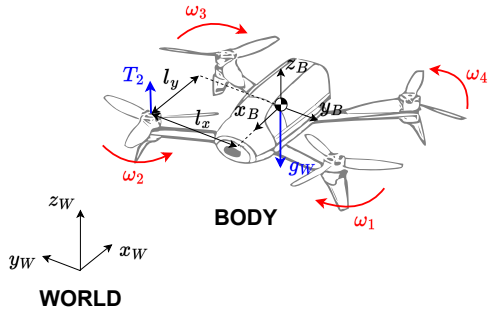


Figure 1: Coordinate System Definitions

3.1 Quadrotor Model

The quadrotor model is based on [17] modelled 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:

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

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, 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_τ the propeller torque coefficient; l_x and l_y are the moment arms shown in Figure 1. All values are provided in Table 1.

3.1.2 Aerodynamic Force Model

The aerodynamic model from [28] captures the major aerodynamic effects in a computationally efficient manner, provided in

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

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 [29]. The symbol $\odot$ represents vector element-wise multiplication.

3.2 Adaptive Neural Controller

The adaptive neural controller design follows closely to the "Neural-Fly" controller architecture 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 is composed of two main components: offline meta-learning and online adaptive control. These two components together build a model of the residual forces acting on a quadrotor of the form:

$$\boldsymbol{f}_{res}(\boldsymbol{x}, \boldsymbol{k}) \approx \phi(\boldsymbol{x}) \boldsymbol{A}(\boldsymbol{k}) \tag{7}$$

where ϕ is a representation function that maps the residual forces dependant 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 ϕ, 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 residual force in Equation 12 we arrive at the control law of the adaptive neural controller:

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

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 $\boldsymbol{u}_f$, a vector of commanded forces from which the corresponding attitude and thrust is obtained through inverse kinematics.

The adaptive neural controller replaces the position controller with the output the desired attitude and thrust commands 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 feedforward 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 residual 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 $\boldsymbol{f}_{res}(\boldsymbol{x}, k)$ 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:

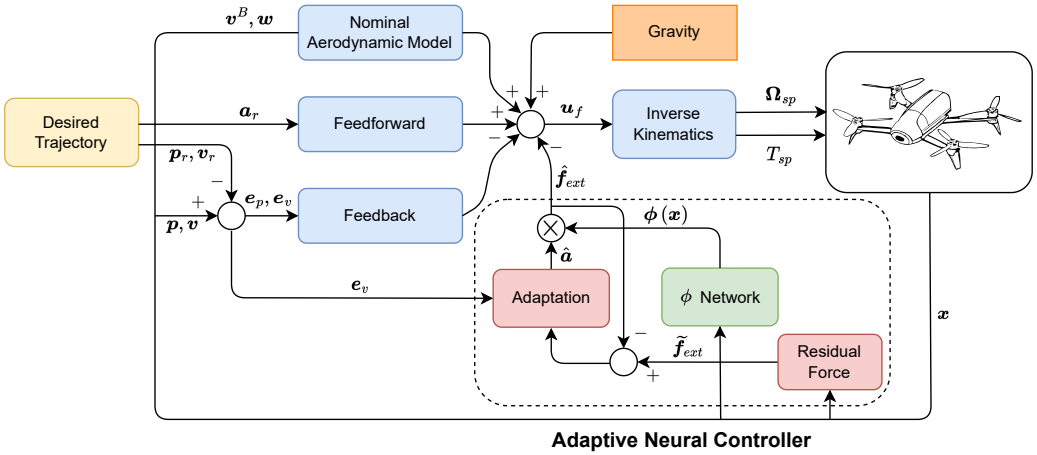


Figure 2: Control Diagram of the Adaptive Neural Controller

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

where K is the total amount of conditions sampled and N_k is the total amount of data-points collected with condition k. The loss 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 are found.

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 over-fitting 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 instead:

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

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 $\alpha \geq 0$. The loss function of h, is typically a cross entropy loss function 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 δ_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 implement 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 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 ultimate goal is to minimize the position tracking error. Thus, the linear coefficients are updated based on a composite error, consisting of the force prediction error and the position tracking error, improving the trajectory tracking performance of the controller. Additionally, the Kalman-filter estimator performs automatic tuning of 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 of 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 outlined 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, has been implemented.

3.3 Trajectory Generation

The desired trajectory of the quadrotor is defined using a spline that minimizes snap, the derivative of acceleration with respect to time $\dot{\boldsymbol{a}}$. Minimum snap trajectories are desirable for quadrotor tracking since motor commands and attitude accelerations are proportional to snap. By minimizing snap these trajectories create a sense of gracefulness of the quadrotors motion, desirable for reducing sensor measurement noise and avoiding excessive control inputs. Additionally, polynomial splines are easily differentiable 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 $\boldsymbol{W}_n \in \mathbb{R}^3$ the desired position of the quadrotor is given by $\boldsymbol{p}_m = \left[p_{m_x}(t), p_{m_y}(t), p_{m_z}(t)\right]$ 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:

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

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 change 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.

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

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. The effect of different time penalty factors on the minimum snap trajectory created is shown in Figure 3.

4 Experimental Setup

The quadrotor platform used is the Parrot Bebop 1, released by Parrot in 2015 as a recreational drone, equipped with a Parrot P7 dual-core Cortex A9 CPU and a MPU6050 IMU. The Parrot Bebop 1 is the workhorse of the Micro Air Vehicle Laboratory (MAVLab) at the faculty of Aerospace Engineering at TU Delft, the platform is well known and

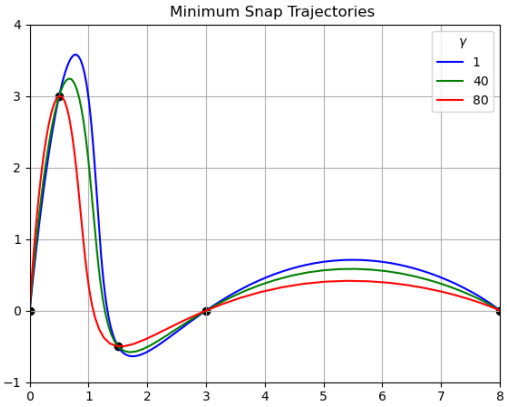


Figure 3: Effect of time factor γ on minimum snap trajectory with $\mu = [1, 40, 80]$ and corresponding total times $T_{tot} = [26.5s, 13.7s, 11.3s]$

readily available. 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 with out-of-the-box rpm sensors.

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_τ	$[Nm^2]$	$6.60e^{-12}$
(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 Parrot Bebop 1 has been flashed with the Paparazzi-UAV open source autopilot project[1] written in C. Designed for 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.

The test flights are performed in the CyberZoo, an indoor $10 \times 10 \times 7$ $[m^3]$ square volume located in the Delft Aerospace Structures and Materials Laboratory. The CyberZoo is equipped with an OptiTrack motion capture system consisting of 8 infrared cameras which provide accurate position, velocity and orientation measurements indoors using the drone's infrared markers. The OptiTrack measurements are fused together 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 4.

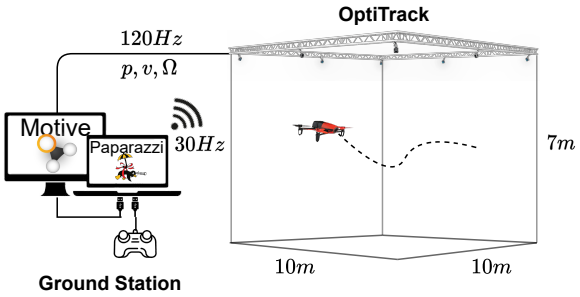


Figure 4: Experimental Setup

5 Results

5.1 Dataset

5.1.1 Data Collection

The DAIML algorithm requires data to train the DNN representation ϕ. 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 exceeds the limits of the CyberZoo a different set of waypoints is sampled till a suitable trajectory is found. All trajectories will travel roughly the same distance but with varying times, resulting in slower trajectories having more datapoints than faster trajectories. To ensure a more uniform distribution of datapoints across velocities multiple trajectories are logged at higher speeds, preventing the network from over-fitting to the most common velocity regime due to it being over-represented in the data.

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 tips increases the wing tip vortices generated by the fast spinning propellers. Resulting in a reduction in propeller lift and increase in propeller drag, altering the aerodynamic effects act-

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

Table 2: Training and Validation Dataset

	Time Factor (μ)	#	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	$1,930$	$61,624$

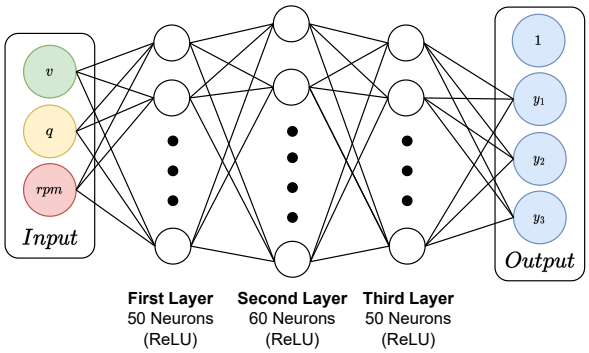


Figure 5: Neural Network $\boldsymbol{\phi}$ Architecture

ing 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 ($4mm$).

5.1.2 Data Processing

The residual forces acting on the quadrotor are obtained by rearranging Equation 1 to solve for f_{res}. The quadrotor acceleration is computed using 1st-order central difference of the velocity measurements. However, the velocity measurements contain significant noise which would be further amplified when calculating the acceleration derivative. To avoid this, a low pass zero-phase "filtfilt" filter is used to smoothen out the velocity measurements, reducing the noise 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 5. 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 in Appendix. C. 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 α influences the degree of adversarial training. A regularization value of $\alpha = 0.1$ was found to strike the right balance between $\boldsymbol{\phi}$ network performance and disentangling the learned linear coefficients into the 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
$\boldsymbol{\phi}$ Network		
Learning Rate	lr_ϕ	$2e^{-3}$
Normalization	γ	17
Training Batch Size	B^t	128
$\boldsymbol{h}$ Network		
Learning Rate	lr_h	$4e^{-4}$
Regularization	α	0.1
Training Frequency	η	0.75
Adaptation Batch Size	B^a	512

The evolution of the $\boldsymbol{\phi}$ network loss given by $\|\boldsymbol{y} - \boldsymbol{\phi}(\boldsymbol{x})\boldsymbol{A}^*\|$ is shown on the training and validation data in Figure 6. The $\boldsymbol{\phi}$ network efficiently learns to minimize this loss with little over-fitting present in the validation loss.

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 co-

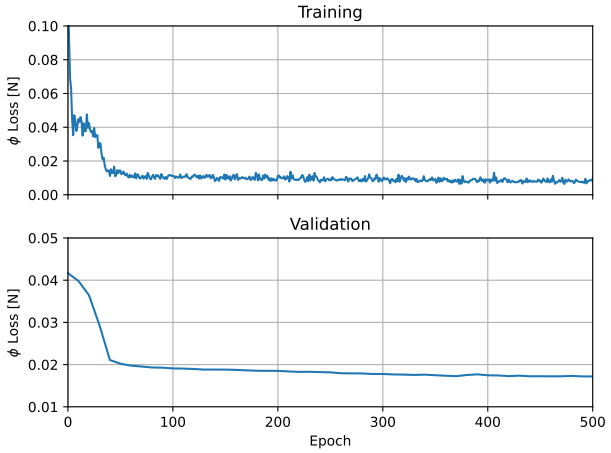


Figure 6: Training and Validation Loss of ϕ Network

efficients into explaining clusters that contain the propeller damage condition information, ensuring the trained representation $\boldsymbol{\phi}$ is condition invariant.

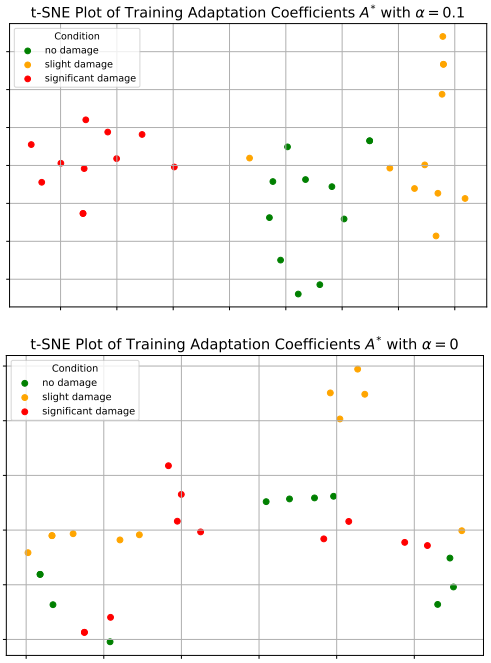


Figure 7: t-SNE plots of the Training Adaptation Coefficients $\boldsymbol{A}^*$ with $\alpha = 0.1$ and $\alpha = 0$

5.2.1 Residual Force Prediction

The residual force predictions of the network $\boldsymbol{\phi}$ are shown on the training, testing and validation data for the no propeller damage condition are shown in Figure 8. The adaptation coefficients learned during training corresponding to the no propeller damage condition are used to compute the unmodeled force predictions $\hat{\boldsymbol{f}}_{ext} = \boldsymbol{\phi}(\boldsymbol{x})\,\boldsymbol{A}^*$.

The $\boldsymbol{\phi}$ network successfully learns a representation that captures the main features of the residual 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 reach 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 $\boldsymbol{\phi}$ network models.

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 due to the fact that a different quadrotor of the same model is used. Substantiating the need of adapting 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 a baseline velocity PID controller. The performance of these controllers is measured tracking a randomized minimum snap trajectory with time factor $\mu = 0.7$ and $\mu = 1.1$, shown in Figure 9 and Figure 13 respectively.

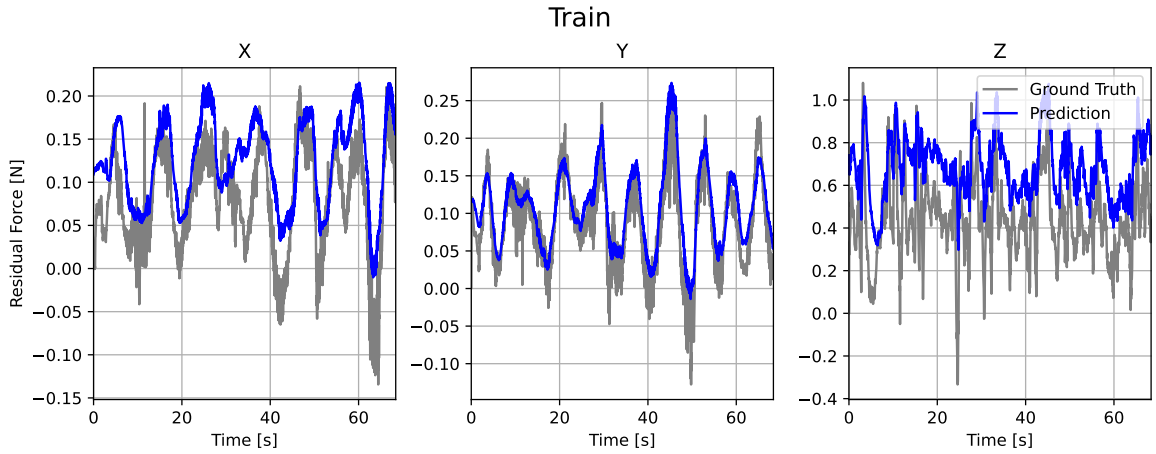
The adaptive neural controller outperforms the other controllers by a substantial margin, achieving a tracking error one order of magnitude lower than the conventional velocity PID controller and approximately $\approx 60\%$ lower than the nonlinear controller. Most importantly, the tracking performance of the adaptive neural controller barely degrades with a less than $3.0\ [cm]$ increase in tracking error with increasing propeller damage. This showcases the ability of the adaptive neural controller at adapting to the current propeller condition flown.

The deterioration of the tracking performance of the adaptive neural controller with increasing speed is shown in Figure 10. Where the average position RMSE for a velocity range is computed for all propeller conditions and both validation minimum snap trajectories $\mu = 0.7$ and $\mu = 1.1$. The tracking error at low speeds is quite high, this is may be caused by the few amount of datapoints at these speeds combined with position errors associated controller initialization. Nevertheless, tracking errors quickly drop to $\approx 0.2\ [m]$ and then slowly increase with increasing speed to about $\approx 0.3\ [m]$ at a maximum velocity of $v_{max} \approx 3\ [m/s]$.

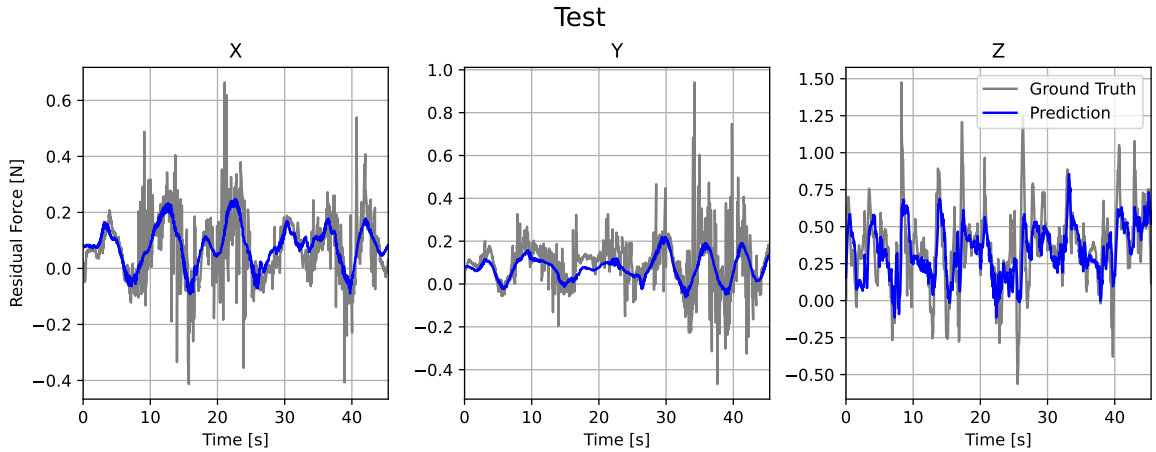
The trajectories flown using the neural adaptive controller and the nonlinear controller are shown in Figure 12 for all propeller damage conditions.

5.4 *Propeller Damage Estimation*

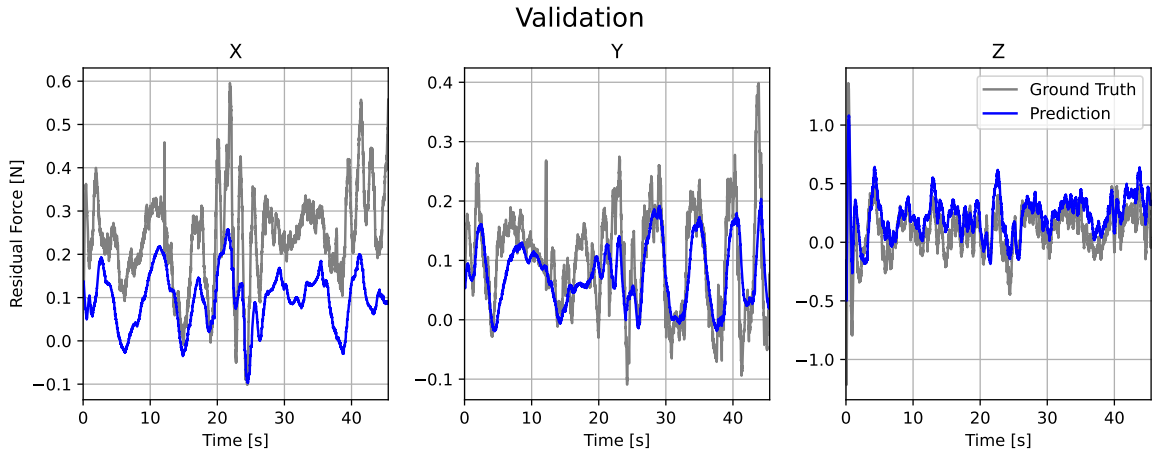
An interesting feature of the adaptive neural controller architecture is the ability to estimate the propeller damage



(a) Training Data



(b) Testing Data



(c) Validation Data

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

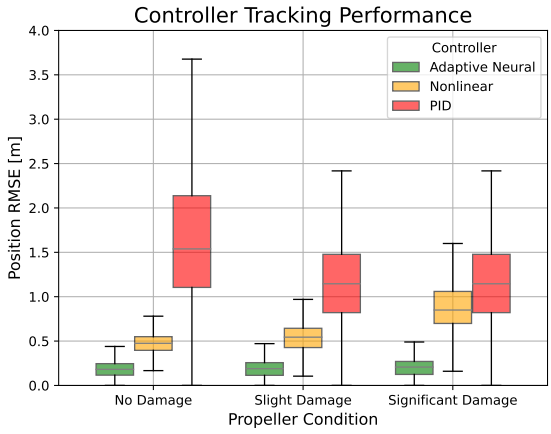


Figure 9: Controller Tracking Performance Comparison on Minimum Snap Trajectory with $\mu = 0.7$

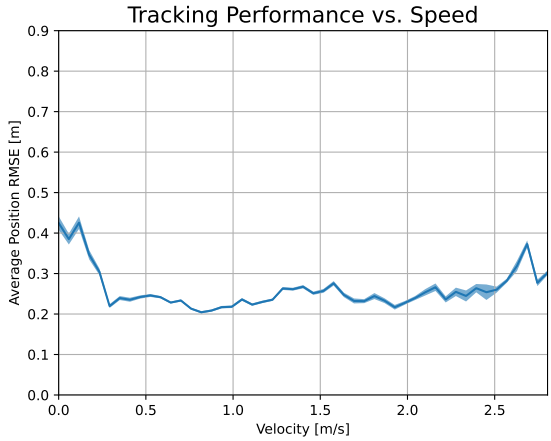


Figure 10: Adaptive Neural Controller Average Position RMSE Bins vs. Speed with 99% Confidence Interval

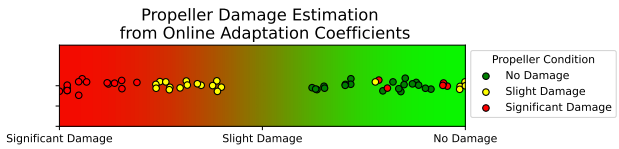


Figure 11: Propeller Damage Condition Estimation from Online Adaptation Coefficients with $\mu = 1.1$

condition from the online adaptation coefficients. The online adaptation coefficients will differ from those in training due to the position tracking term in the coefficient update in Equation 12, nevertheless sufficient propeller condition information should still remain. To showcase the feasibility of estimating the propeller damage condition from the online adaptation coefficient, t-SNE is performed on the higher dimension adaptation coefficients to reduce them into one dimension from which the propeller damage can be easily estimated as shown in Figure 11.

6 Discussion & Recommendations

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

6.1 *Computational Efficiency*

Most quadrotor adaptive controllers in literature require powerfull 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 the 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 $\approx\ 3m$ to train a network from scratch on a laptop.

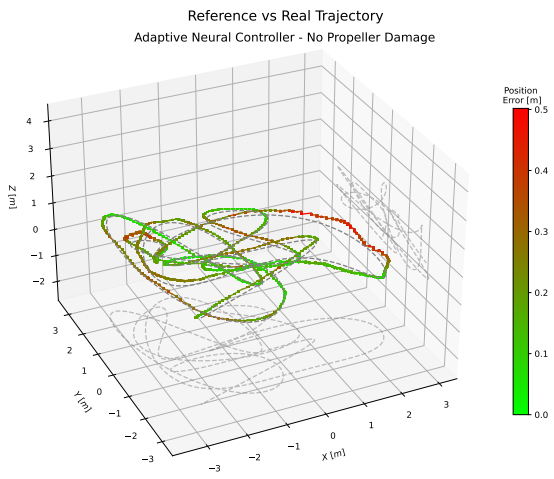
6.2 *Influence of Propeller Damage*

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 alternatively 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.

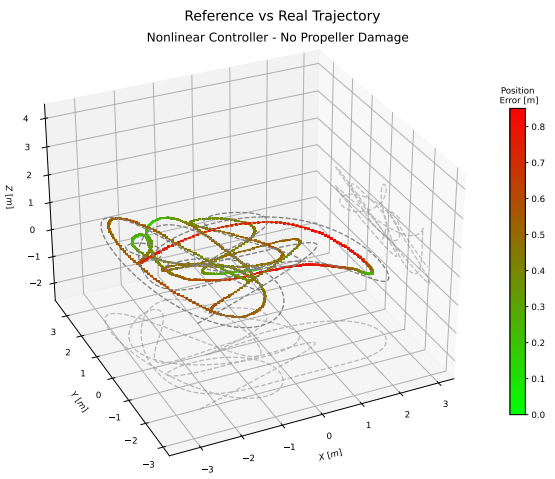
6.3 *Noisy Measurements*

The ϕ 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 ϕ.

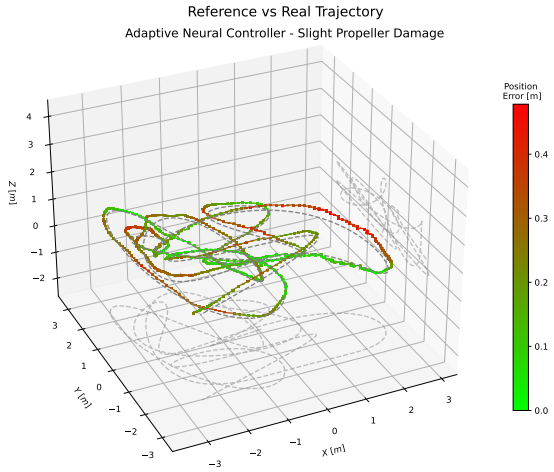
The poor quality of the OptiTrack measurements also influences the online adaptation. Significant filtering of the velocity measurements is required before applying backward finite difference to obtain usable measurements of the residual force $\boldsymbol{y}$ which have significant lag. This may cause oscillatory behaviour as can be seen in some of the trajectories in Figure 12.



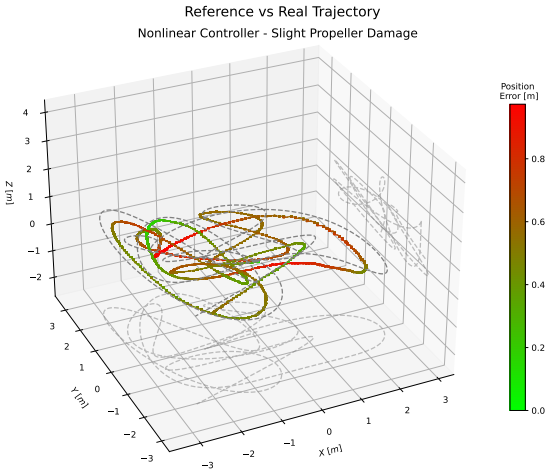
(a) Adaptive Neural Controller w/ No Propeller Damage



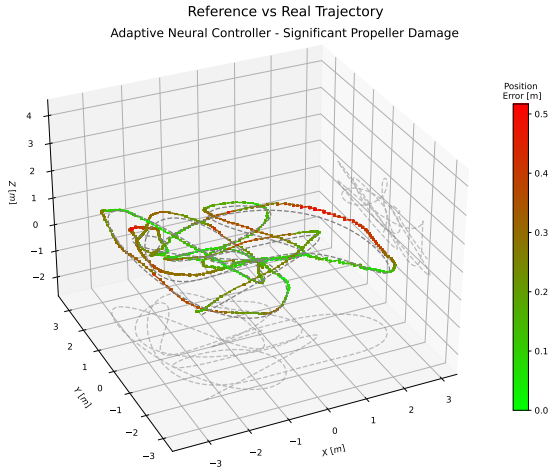
(b) Nonlinear Controller w/ No Propeller Damage



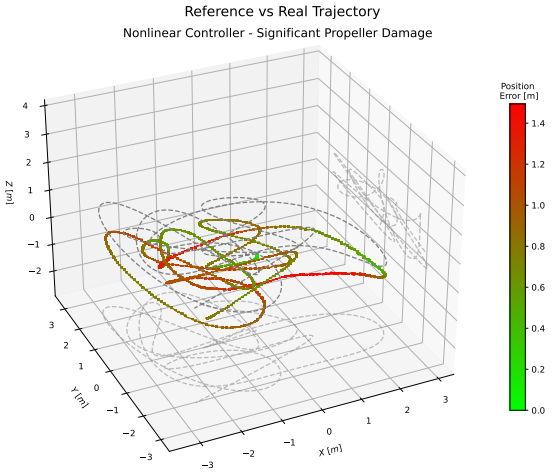
(c) Adaptive Neural Controller w/ Slight Propeller Damage



(d) Nonlinear Controller w/ Slight Propeller Damage



(e) Adaptive Neural Controller w/ Significant Propeller Damage



(f) Nonlinear Controller w/ Significant Propeller Damage

Figure 12: Comparison of Trajectories flown using Adaptive Neural Controller and Nonlinear Controller for all Propeller Conditions

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 [30] 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.

7 Conclusion

The "Neural-Fly" controller from [27] originally applied to wind speed conditions has been investigated further. Neural-Fly is a lightweight yet powerful adaptive neural controller that combined pre-trained condition invariant representations of the unmodeled forces using Domain Adversarially Invariant Meta-Learning (DAIML) with fast online adaptation to the current condition. 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 at 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 of attaching 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 ϕ 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.

Acknowledgements

The author is grateful to Erik van der Horst for his guidance & technical expertise and to my parents for their unconditional love.

References

[1] NASA's Ingenuity Mars Helicopter Successfully Completes First Flight.

[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. arXiv:1903.02091 [math].

[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 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. arXiv:1712.02402 [cs].

[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. arXiv, February 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. arXiv:1910.07730 [math].

[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. arXiv:2109.04210 [cs].

[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. arXiv:2105.13527 [cs, eess].

[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. 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. arXiv:2004.11345 [cs].

[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] 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, 2017.

[29] 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.

[30] Drew A. Hudson and C. Lawrence Zitnick. Generative Adversarial Transformers, March 2022. arXiv:2103.01209 [cs].

Appendix A: Controller Performance Comparison

The tracking performance of the neural adaptive, nonlinear and PID controller on a minimum snap trajectory with time factor $\mu = 1.1$ is shown in Figure 13.

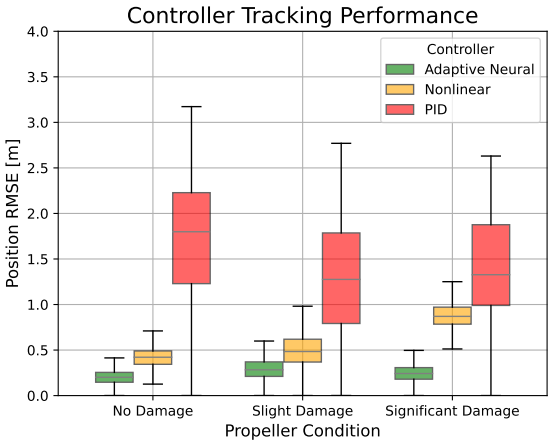
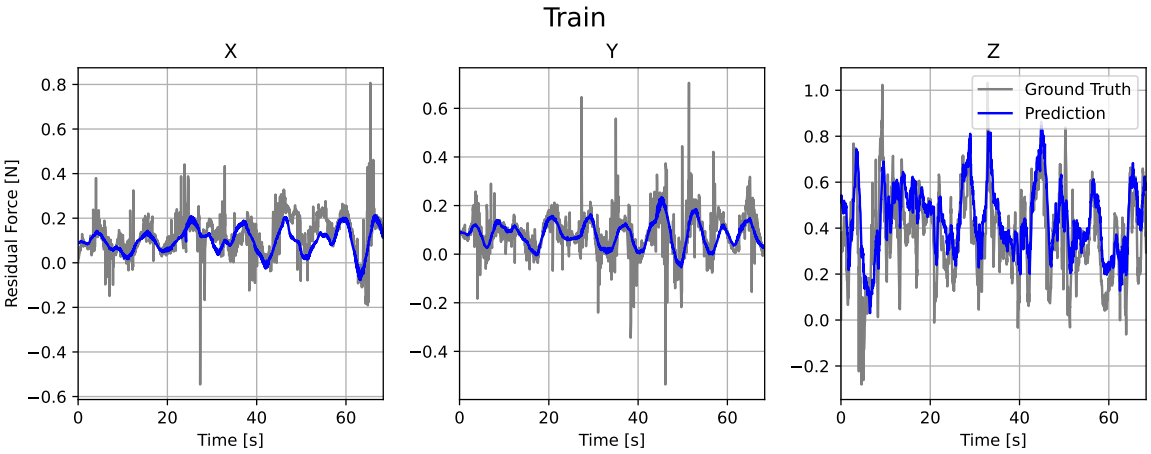


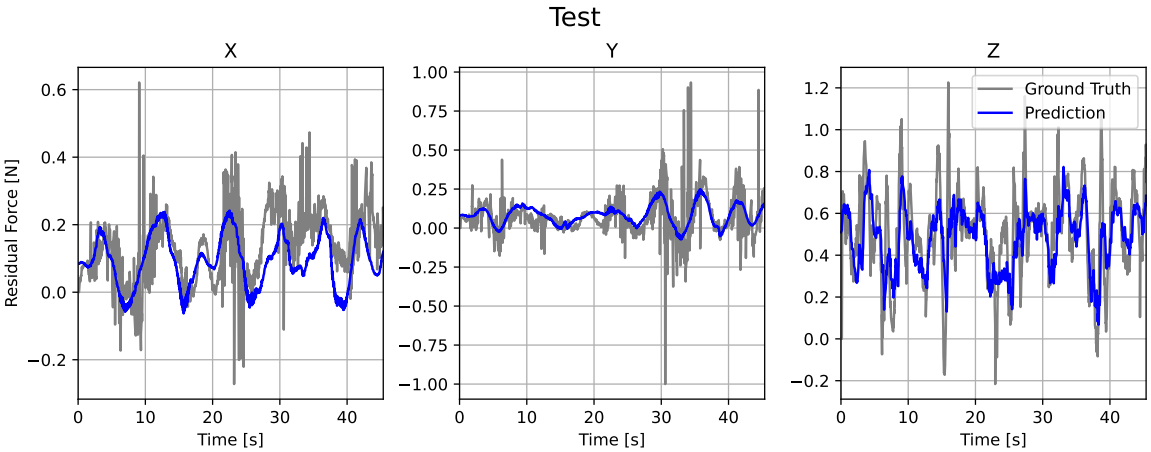
Figure 13: Controller Tracking Performance Comparison on Minimum Snap Trajectory with $\mu = 1.1$

Appendix B: Residual Force Predictions

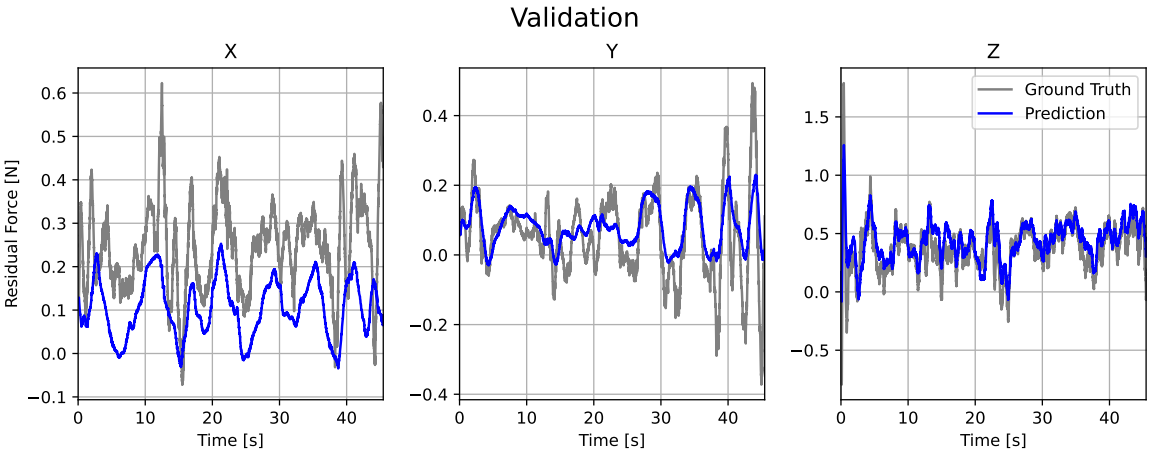
The residual force predictions for the slight propeller damage and significant damage condition on the training, testing and validation data are shown in Figure 14 and Figure 15 respectively.



(a) Training Data

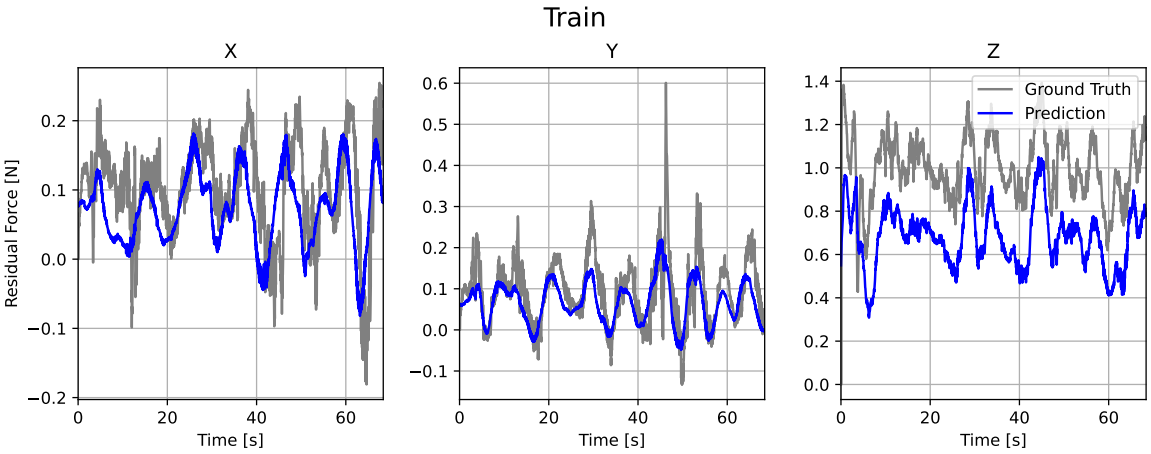


(b) Testing Data

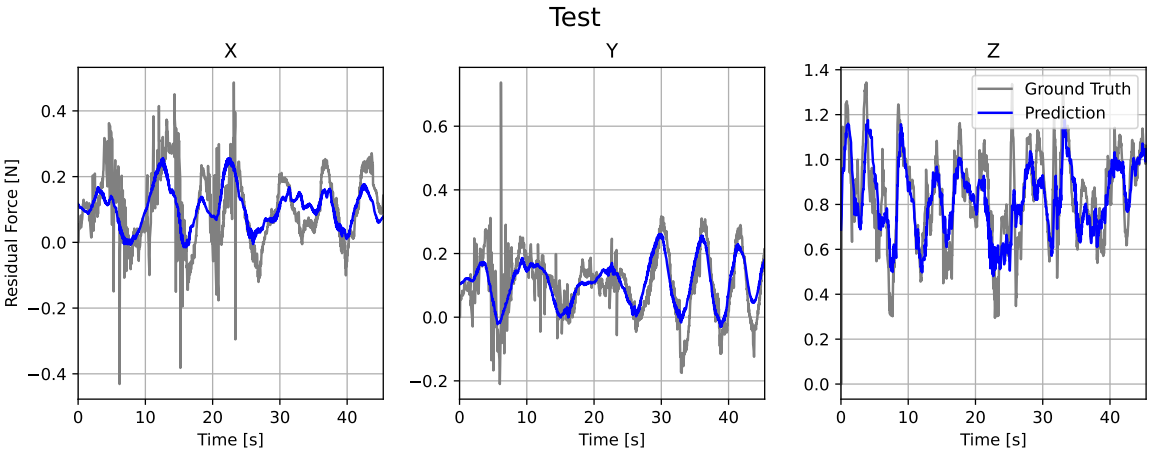


(c) Validation Data

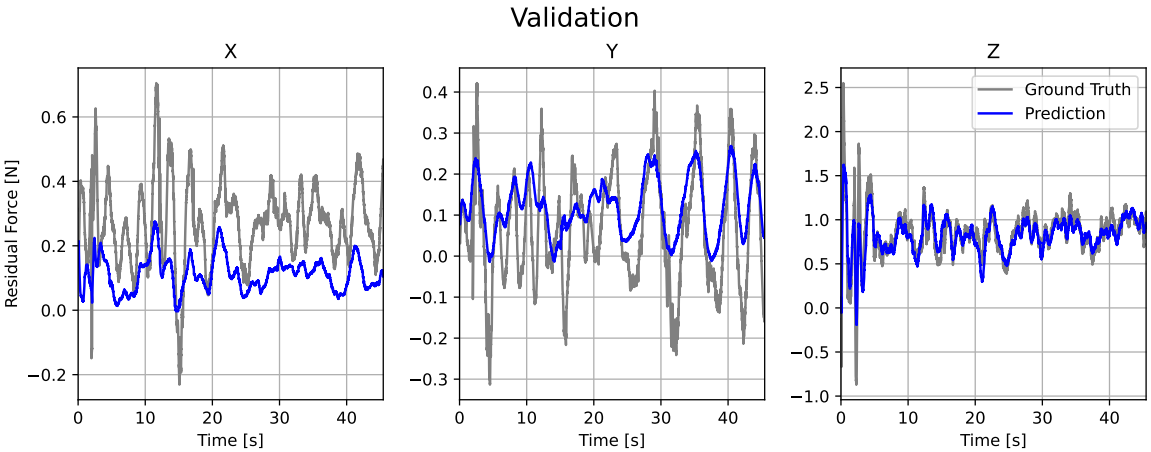
Figure 14: Residual Force Predictions $\hat{\boldsymbol{f}}_{ext} = \boldsymbol{\phi}(\boldsymbol{x})\,\boldsymbol{A}^*$ for Slight Propeller Damage Condition



(a) Training Data



(b) Testing Data



(c) Validation Data

Figure 15: Residual Force Predictions $\hat{\boldsymbol{f}}_{ext} = \boldsymbol{\phi}(\boldsymbol{x})\,\boldsymbol{A}^*$ for Significant Propeller Damage Condition

APPENDIX C: SENSITIVITY ANALYSIS

A sensitivity analysis is performed on DAIML to determine the optimal hyperparameters shown in Table 3 using Bayesian search that minimize the best test loss during training. The hyperparameters searched are given in Table 4.

Table 4: Sensitivity Analysis Hyperparameter Value Range

Hyperparameter	Symbol	Values		
		Min	Max	Distribution
Spectral Norm	SN	$False$	$True$	Categorical
Normalization	γ	5	20	Int Uniform
Training Frequency	η	0.2	0.8	Uniform
ϕ Learning Rate	lr_ϕ	$1e^{-4}$	$1e^{-3}$	Uniform
h Learning Rate	lr_h	$5e^{-4}$	$5e^{-3}$	Uniform

Figure 17: Sensitivity Analysis Hyperparameter Importance & Correlation

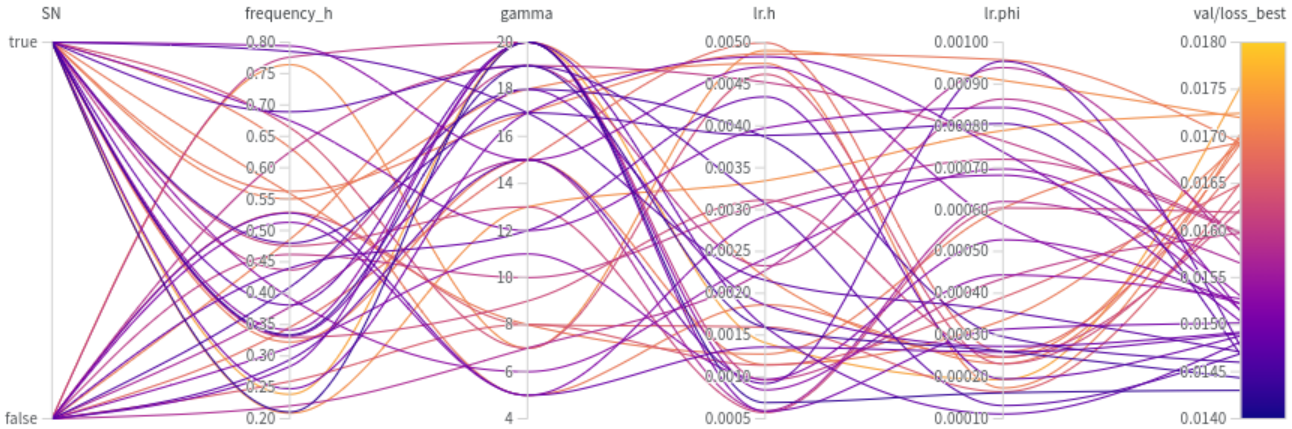


Figure 16: Sensitivity Analysis

Part II
Literature Review

Nomenclature

Abbreviations

Abbreviation	Definition
CPC	Complementary Progress Constraint
COBYLA	Constrained Optimization BY Linear Approximation
DFBC	Differential-Flatness Based Control
(D)NN	(Deep) Neural Network
IMU	Inertial Measurement Unit
INDI	Incremental Nonlinear Dynamic Inversion
LQR	Linear-Quadratic Regulator
MIAC	Model Identification Adaptive Controllers
MPC	Model Predictive Controller
MRAC	Model Reference Adaptive Controllers
NDI	Nonlinear Dynamic Inversion
NMPC	Nonlinear Model Predictive Control
OCP	Optimal Control Problem
PID	Proportional-Integral-Derivative
RPM	Rotations Per Minute
RMS(E)	Root Mean Squared (Error)
SGD	Stochastic Gradient Descent
SoC	System on a Chip
t-SNE	t-Distributed Stochastic Neighbor Embedding
UAV	Unmanned Aerial Vehicle

Symbols

Symbol	Description	Unit
α	Angle of Attack	$[Rad]$
θ	Pitch Angle	$[Rad]$
ϕ	Roll Angle	$[Rad]$
ψ	Yaw Angle	$[Rad]$
p	Roll Rate	$[Rad/s]$
q	Pitch Rate	$[Rad/s]$
r	Yaw Rate	$[Rad/s]$
$\boldsymbol{q}$	Quaternion Orientation	$[-]$
$\boldsymbol{f}_{res}$	Residual Force Vector	$[N]$
$\boldsymbol{f}_{a}$	Aerodynamic Force Vector	$[N]$
$\boldsymbol{f}_{t}$	Thrust Force Vector	$[N]$
$\boldsymbol{p}$	Position Vector	$[m]$
$\boldsymbol{v}$	Velocity Vector	$[m/s]$
$\boldsymbol{a}$	Acceleration Vector	$[m/s^2]$
$\boldsymbol{\tau}$	Torque Vector	$[Nm]$
$\boldsymbol{\Omega}$	Angular Velocities Vector	$[Rad/s]$
$\boldsymbol{\alpha}$	Angular Acceleration Vector	$[Rad/s^2]$
$\boldsymbol{I}$	Quadrotor Mass Moment of Inertia Matrix	$[kg \cdot m^2]$
I_r	Rotor Inertia	$[kg \cdot m^2]$
$\boldsymbol{x}$	Residual Force Measurement Vector	$[N]$
$\boldsymbol{y}$	Phi Network Input Vector	$[N]$
K	Controller Gain	$[-]$
	Constants	
$\boldsymbol{g}$	Gravitational Acceleration Vector	$[0, 0, 9.81\ m/s^2]$
m	Parrot Bebop 1 Mass	$0.390[kg]$

List of Figures

List of Tables

Abstract

Executing fast & precise trajectories with quadrotors in challenging environments is essential for applications such as: search & rescue, aerial delivery/transport, autonomous drone racing or even space exploration. Complex aerodynamic disturbances arise during agile high-acceleration flight introducing large tracking errors, rendering traditional controller designs insufficient. State-of-the-art trajectory tracking controllers achieve impressive tracking performances, but require highly accurate models to achieve these results and require powerful embedded computers to run onboard. Modeling uncertainties arising from varying payloads, propeller damage or parameter mismatch significantly degrade controller performance. Adaptive controllers augment existing controllers, they learn and compensate for these model uncertainties but crucially suffer from stability and excitation problems and require additional computational resources, hindering their wider commercial adoption. Recent work on Adaptive Deep Neural Network based control has shown impressive results adapting to high wind conditions, as a computationally light-weight alternative with comparable performance to state-of-the-art adaptive methods. This literature review aims to shed light into the largely unexplored field of adaptive neural controllers, providing the necessary groundwork to improve the Adaptive Neural Controller architecture and perhaps adapting it to different conditions.

1

Introduction

Autonomous Unmanned Aerial Vehicles (UAVs) are a disruptive technology that over the next decades will revolutionize the agriculture, surveillance, transport, search & rescue industries to name a few. The UAV market revenue size was valued at $30 Billion in 2020 and is project to reach $55 billion by 2030 according to DroneII [9], showing no signs of slowing down. Autonomous UAVs have already been used in a wide range of applications from performing the first controlled flight on another planet [24] to delivering life-saving medical supplies to remote African regions [42].

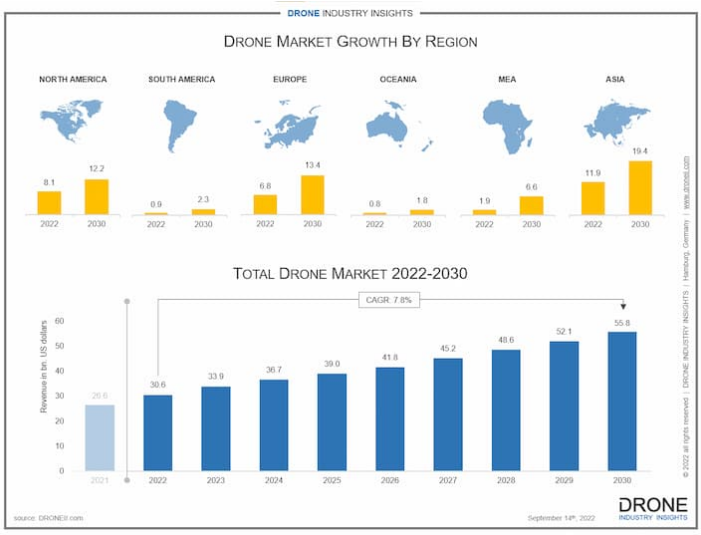


Figure 1.1: UAV Market Revenue Forecast [9]

Accurate trajectory tracking of quadrotors, a common type of UAV with four rotors, at high-speeds is a challenging problem. Due to the complex relationship between aerodynamic forces and quadrotor maneuverability at these speeds designing a controller using traditional control methods results in significant tracking errors. Moreover, changes in conditions such as wind, propeller damage or payloads is a common occurrence in real-world situations which significantly degrades the tracking performance of controllers. However developing an accurate & light-weight adaptive trajectory tracking controller is an essential step in the wider commoditization of UAVs. Allowing quadrotors to execute trajectories accurately and therefore safely in a wide range of conditions is essential for applications such as search & rescue, aerial delivery/transport, space exploration and autonomous drone racing applications.

Figure 1.2: Drone Flying in Windy Conditions at the Caltech Real Weather Wind Tunnel [25]

Figure 1.3: Drone Flying with a Suspended Payload [31]

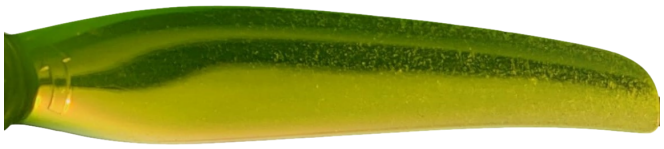


Figure 1.4: Propeller Damage from Flying $\approx 20\ [m]$ in Abrasive Particles

Conventional Proportional-Integral-Derivative (PID) controllers use a cascaded controller architecture, while simple yet effective at low speeds their accuracy quickly degrades at higher speeds. State-of-the-art trajectory tracking controllers such as Nonlinear Model Predictive Control (NMPC) or Differential Flatness Based Control (DFBC) achieve impressive tracking performances at high speeds with a Root-Mean Squared (RMS) tracking error of $8.2\ [cm]$ at speeds of up to $20\ [m/s]$ [36], but require powerful embedded computers to run onboard and rely on highly accurate models to achieve these results.

Adaptive controllers augment existing controllers by either estimating the values of unknown coefficients online or using an $\mathcal{L}_1$ adaptive controller to directly estimate the observable aerodynamic residual force. However, online coefficient estimation suffer from convergence problems when there is a lack of excitation and must perform a trade-off between lag and noise in the residual force signal. Additionally, adaptive controllers add to the computational complexity of the controller further straining the limited computational budget available onboard.

Recent work on ”Neural-Fly”, a lightweight controller combining pre-trained representations trough Deep Neural Networks (DNNs) with rapid online adaptation achieves impressive results. Neural-Fly outperforms state-of-the-art adaptive controllers in challenging wind conditions, achieving a RMS tracking error of $9.4\ [cm]$ at wind speeds of $12.1\ [m/s]$ [25]. However this adaptive neural architecture is largely unexplored lacking reproducibility, it is unknown how well these results would translate to other drones or possibly adapting to different conditions.

The main challenge of developing an accurate trajectory tracking controller for aggressive quadrotor flight is to perform an applicable trade-off between model accuracy and complexity. Identifying a model capable of describing the complex aerodynamic effects at these high speeds while remaining tractable to run the control loop at a high-frequency. The lack of onboard computational resources is further compounded by the fact that autonomous applications devote most of the available computational budget onboard towards image processing, rendering the implementation of most of these state-of-the-art controllers impractical for autonomous applications.

The aim of this thesis is to investigate adaptive neural controllers as an accurate yet computationally efficient adaptive controller architecture for high-speed trajectory tracking in varying conditions such as wind, propeller damage or payloads. In this literature review the groundwork has been laid, starting with

a literature review on the progress of trajectory tracking controllers in Chapter 2. Followed in Chapter 3 by an explanation of relevant research questions and research objectives related to developing such a controller. In Chapter 4 the methodologies regarding the trajectory planning and controller has been outlined. The data collection & processing procedure has been explained in Chapter 5 while the training results of the DAIML algorithm has been analyzed in Chapter 6. Lastly, the results, outcome & relevance of this thesis has been discussed in Chapter 8.

2

Literature Review

The literature review has been divided into 3 main sections. First, the progress of conventional quadrotor trajectory tracking controllers is presented in Section 2.1 and aerodynamic models in Section 2.1.1. Followed by an explanation of adaptive trajectory tracking controllers in Section 2.2. Lastly, a review of adaptive neural controllers is included in Section 2.3.

2.1. Quadrotor Trajectory Tracking Control

Quadrotors are inherently unstable platforms. Initial work on quadrotor control achieved stable hover and near-hover flight thanks to small-angle assumptions allowing for conventional linear control methods such as cascaded Proportional-Integral-Derivative (PID) [18] or Linear-Quadratic Regulator (LQR) [18] to be implemented. These methods are simple yet effective in low-speed regimes, however their performance quickly degrades in high-speed agile flight where the small angle assumptions no longer hold. Despite these drawbacks cascaded PID controllers are widely used in off-the-shelf flight controllers, due to ease of tuning and modular design, splitting the controller into position/velocity/attitude/rate controllers and a thrust mixer as shown in Figure 2.1.

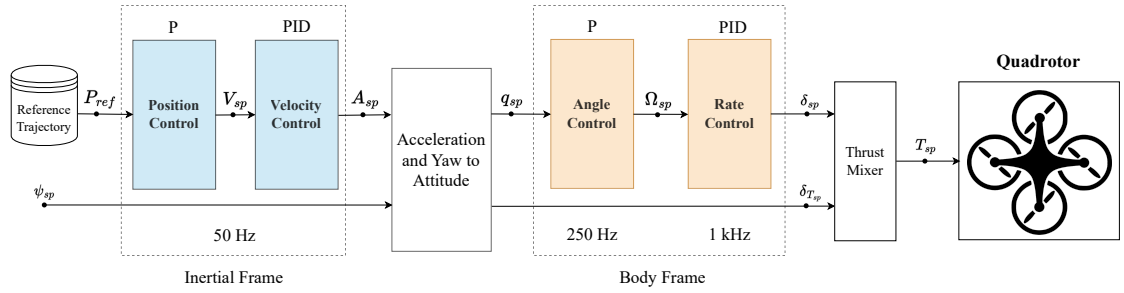


Figure 2.1: Quadrotor PID Control Architecture

Non-linearities from the attitude dynamics render the small angle assumption no longer valid for agile flight. Nonlinear flight controllers have been proposed to solve this problem, such as Nonlinear Dynamic Inversion (NDI) [34], also referred to as feedback linearization, which enables "the use of a linear control law by transforming the nonlinear dynamics into a linear input-output map" [38]. However, exact dynamic inversion suffers from a lack of robustness. Variants of NDI have been proposed to tackle this issue, such as backstepping design [13] which recursively designs a controller starting from a known stable inner system and "back out" progressively stabilizing the entire system. "More recently an incremental version of NDI (INDI) [41] has been proposed, which provides robustness by incrementally applying control inputs based on inertial measurements" [38].

State-of-the-art trajectory tracking controllers can be categorized into: non-predictive and predictive methods. Non-predictive methods track a single reference step, whereas predictive methods encode several future timesteps into the control command.

DFBC is a non-predictive controller, which takes advantage of the fact that quadrotors are differentially flat systems [10], enabling the reformulation of the trajectory tracking problem as a state tracking problem. The tracking performance of DFBCs is improved by using an inner loop INDI controller and aerodynamic model as shown in Figure 2.2, achieving a RMS tracking error of $12.2\ [cm]$ at a speed of $20\ [m/s]$ [38].

On the other hand, NMPC is a predictive method, which generates motor commands in a receding horizon fashion, subject to solving a constrained optimization problem on the tracking error over the predicted time horizon. Similarly, the performance of NMPCs is bolstered using an INDI rate controller and aerodynamic model. NMPCs outperform DFBCs as they minimize the tracking error over multiple timesteps whereas DFBCs are too short-sighted only considering one reference point achieving a RMS tracking error of $10.2\ [cm]$ at a maximum speed of $20\ [m/s]$ [39]. However, solving this nonlinear optimization problem over multiple time-steps come at the cost of significantly higher computational power in the order of 100 times higher than DFBCs as shown in Figure 2.3. Crucially both of these controllers require highly accurate quadrotor and aerodynamic models to achieve these results. Modelling mismatch will occur in real-world conditions caused by either wind, propeller damage or payloads significantly degrading controller performance. A reduction in the thrust coefficient by 30% will result in a significant degradation in controller performance, with position RMSE rising from $0.08\ [m]$ to $1.36\ [m]$ for NMPC and from $0.09\ [m]$ to $1.01\ [m]$ for DFBC [36].

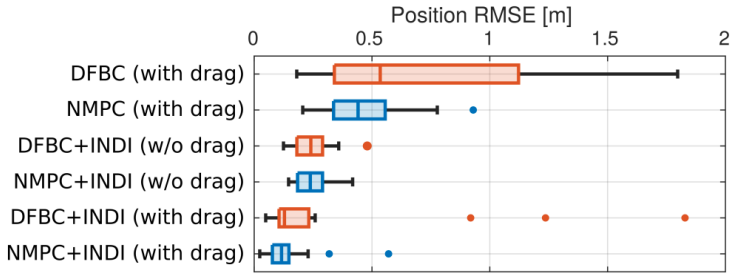


Figure 2.2: Box plots of the position tracking root-mean-square-error tracking reference trajectories with speeds up to $20\ m/s$ [36]

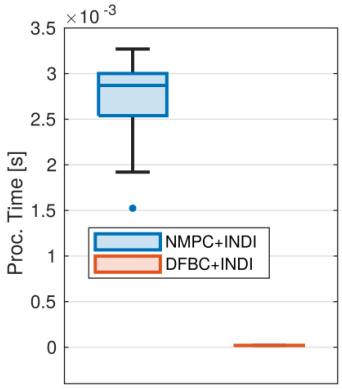


Figure 2.3: Processing time to generate control commands [36]

2.1.1. Aerodynamic Model

Operating a quadrotor at high speeds and through agile high-acceleration maneuvers involves complex aerodynamic disturbances that consequently introduce large tracking errors. "These effects are difficult to model, since they consist of a combination of propeller lift and drag dependent on the induced airstream velocity, fuselage drag, and complex or even turbulent effects due to the interaction between the propellers, the downwash of other propellers, and the fuselage" [39]. Flying with varying conditions such as wind, propeller damage or suspended payloads further compounds the complexity of these aerodynamic effects given the unsteady aerodynamic interactions caused by the induced airspeed, rotors and payloads.

Most of the trajectory tracking controllers discussed improve tracking performance by compensating for these aerodynamic effects. A common approach partially captures these aerodynamic effects using simple linear of quadratic drag models [37] obtained using system identification techniques from external measurements. These models are effective at lower speeds where the complex aerodynamic disturbances are not dominant and cannot adapt to changes in the aerodynamic model caused by

varying conditions. While higher fidelity aerodynamic model can be derived from Computational Fluid Dynamics (CFD) simulations [40], they require large compute clusters and time-consuming platform-specific meshing. Ultimately, CFD results must be interpolated to be computationally tractable onboard at the required frequencies of the control loop.

2.2. Adaptive Control

Adaptive control consists of continuously monitoring the systems behaviour and updating the control strategy accordingly to maintain optimal performance. Adaptive control of systems with parametric uncertainty has been extensively researched. Adaptive controllers are often an augmentation to existing controllers rather than standalone controllers, in the context of UAV control the two dominant adaptive control architectures are Model Identification Adaptive Controllers (MIACs) and Model Reference Adaptive Controllers (MRACs).

MRACs achieve the desired behaviour of the system by continuously adapting the controller parameters, typically using an $\mathcal{L}_1$ adaptive controller, comparing the systems output with that of a reference model. Ensuring that the controlled system closely follows the desired reference model's response in the presence of disturbances or uncertainties. MRACs have been successfully deployed for quadrotor trajectory tracking, achieving slight tracking performance degradation with suspended weights attached [14, 19]. Albeit, both of these implementations require significant computational resources either required a ground station in-the-loop [19] or attaching an NVIDIA Jetson TX2 [14] and in the case of [19] is limited to tracking simple circular trajectories not exceeding $2\ [m/s]$. Furthermore, MRACs are susceptible to noise in the system output measurements requiring a trade-off in the filter design between noise and lag.

MIACs utilize system identification techniques to estimate or learn the mathematical model of a system online and adapting controller parameters accordingly. The mathematical model and corresponding basis functions may be physics based or random Fourier feature based. Physics based modeling is the more logical option, Physics based MIAC is used in [33] for aerodynamic coefficient estimation using an airflow sensor of a VTOL aircraft, achieving hover flight with wind speeds of $9\ [m/s]$. However, physic based modeling requires extensive knowledge of mathematical models describing system and suffers from convergence issues when there is a lack of excitation. On the other hand, random Fourier feature based modeling does not require any prior knowledge of the system but it is tricky to determine the necessary frequencies that describe the system efficiently in the high dimensional Fourier feature space. Nevertheless, random Fourier based MIACs are deployed in [3] to adapt to wind conditions with a drag plate attached, outperforming a MRAC using $\mathcal{L}_1$ adaptive control.

2.3. Adaptive Neural Control

Neural network based adaptive control has been extensively researched [5, 22] but these networks where limited to being too shallow and lacked efficient pre-training mechanisms. Early approaches used NNs in MRACs to model the unknown system and generate the feedback control, updating the weights based on the error between system output and model output [6]. Recent work in the field of deep-learning-based trajectory tracking controllers, that better utilize the representation power of multi-layered Deep Neural-Networks (DNN), has shown promising results. In [16], a combination of temporal convolutions and a three-layered network is used to perform challenging acrobatic manoeuvres with a tracking error of $24\ [cm]$ at speeds of $4.5\ [m/s]$, albeit they also run into computational load issues with their network running at $100\ Hz$ on an NVIDIA Jetson TX2.

An important goal in autonomy is to enable autonomous robots to learn from prior experience and quickly adapt to new tasks or environments, so called "meta-learning" or "learning-to-lean". Meta-learning is a field of research which aims to learn an efficient model from data collected in various conditions. "The learned model, typically represented as a DNN, ideally should be capable of rapid adaptation to unseen environments/tasks" [25]. Meta-Learning has shown great potential in the field of robotics, enabling online dynamics model identification of highly-dynamic systems even in unknown environments. For example, meta-learning has enabled the quick adaptation of legged millirobots to

unseen terrain [21] and to varying suspended payloads from drones [2].

Impressive results have been achieved in [25] with a four-layered network, called Neural-Fly, achieving state-of-the-art tracking performance of $9.4\ [cm]$ at wind speeds of $12.1\ [m/s]$ running onboard a less powerful Raspberry Pi 4 computer at $100\ [Hz]$. The Neural-Fly controller can be divided into two main components: the offline learning phase and the online adaptive control phase. The offline learning phase consists of using a novel Domain Adversarially Invariant Meta-Learning (DAIML) algorithm to learn a wind condition independent DNN representation of the aerodynamics acting on the quadrotor from as little as 12 minutes of flight data. The output of the DNN is then combined with a set of linear coefficients from a composite online adaptive controller which combines the position and aerodynamic force prediction errors to finally obtain the estimate of the aerodynamic forces. With this architecture Neural-Fly demonstrates state-of-the-art tracking performance with an average improvement of $\approx 35\%$ over an INDI controller, without the need of additional sensors or powerful embedded computers" [25].

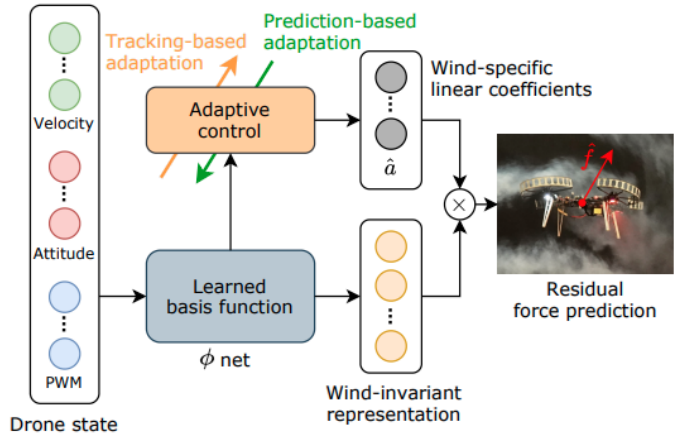


Figure 2.4: Online adaptation block where the basis function is adapted online with the tracking & prediction based error to obtain the force estimate $\hat{f}_{ext} = \phi\hat{a}$ [25]

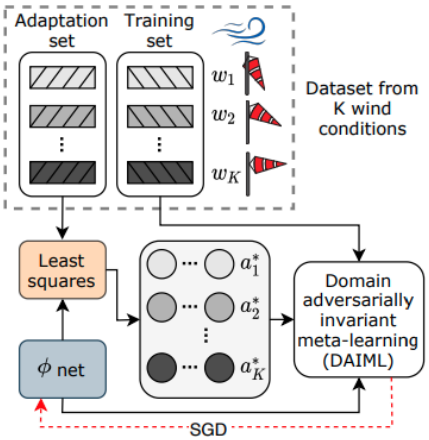


Figure 2.5: Illustration of the meta-learning algorithm DAIML on data collected from wind conditions $\{w_1, \cdots, w_K\}$ [25]

While impressive results have been achieved with Neural-Fly on significantly less powerfull embedded hardware, these test are limited to being done in high-wind conditions instead of propeller damage, it is unknown wherever this performance could translate to high-speed flight.

3

Research Questions

This section consists of two parts: the research question & sub-questions in Section 3.1 and the research objective & sub-goals in Section 3.2 .

3.1. Research Question

Based on the literature review in Chapter 2 the following research question is proposed:

> How well does an Adaptive Neural Network controller perform at quadrotor trajectory tracking with various propeller damage conditions in terms of tracking error and computational complexity?

The main research question raises several sub-questions explained below:

1. **How well does Neural-Fly controller architecture perform adapting to propeller damage in high-speed flight and (if applicable) what areas can be improved?**
   The Neural-Fly controller architecture was designed with varying wind-speed conditions in-mind, various changes mey be necessary to re-design the controller for propeller damage conditions. Additionally, a sensitivity analysis will be performed on the important design choices of the controller to identify potential areas of improvement. These improvements will then be applied to enhance the performance of the controller in terms of tracking error or computational complexity.
2. **Can close-loop stability and robustness guarantees be provided?**
   Guaranteeing closed-loop stability and robustness guarantees is a difficult task when designing a DNN-based controller, however it is an important safety consideration to ensure that the controller does not generate unpredictable outputs that lead to close-loop instability. Thanks to the innovative split architecture of the Neural-Fly controller into a pre-trained network and a relatively small online adaptation, stability and robustness guarantees can be proven mathematically.
3. **How well does the controller perform in unseen conditions?**
   An important challenge of any machine learning project, specially one used in a quadrotor controller, is to design a controller that can extrapolate to speeds or trajectories which it has not training on, so called ”unseen conditions”. The performance of the controller in these unseen conditions will be investigated and suggestions could be provided to reduce the performance degradation.
4. **How can the performance of the controller be validated and compared to other state-of-the-art controllers?**
   There is no standardized testing procedure of trajectory tracking controllers which in addition to the extensive amount of variables influencing the performance of the controller leads to widely varying results across different research papers. To ensure that the results obtained are comparable to those from other research papers a testing procedure will be designed that is repeatable and comparable. Additionally, other controllers will be tested in this testing procedure if possible.

3.2. Research Objective

The main research objective of this thesis is:

> To deploy an Adaptive Deep-Neural-Network controller for quadrotor trajectory tracking with comparable performance to state-of-the-art controllers and lower computational complexity with varying propeller damage conditions.

Achieving this research objective is a complex & challenging task, to aid in this the main objective has been divided into smaller more manageable sub-tasks that have been outlined in chronological order below:

1. **Data Collection & System Identification**
   The first step consists of collecting real-world data which typically consists of logging the drone's velocity, orientation and motor speeds while the drone tracks a trajectory with varying propeller damage conditions. The data is then processed to calculate the unmodeled forces acting on the quadrotor which are obtained by comparing the expected forces from an identified drone model or from an existing model to the measured forces.
2. **Offline-Learning & Online Adaptation**
   In the offline learning phase, the inputs: relevant drone states and the output: the residual forces are fed into a DNN which learns a propeller damage invariant representation of the unmodeled forces using the DAIML algorithm. The online adaptation phase involves fast online adaptation to the current propeller damage conditions based on the composite position and force prediction errors. The output of the DNN is combined with online adaptation coefficients to hopefully obtain an unfiltered zero-lag prediction of the residual force, which is then compensated for by the trajectory tracking controller.
3. **Design Sensitivity Analysis & Improvement Implementation**
   The design sensitivity analysis consists of varying key design choices of the controller and measuring their impact on the performance of the controller in terms of tracking error and computational complexity. For example, this could involve a hyper-parameter search of the DNN architecture or changing the input drone states of the network. The results of the sensitivity analysis can then be used to identify improvements and implement them.
4. **Unseen Conditions Analysis**
   The ability of the controller to extrapolate to speeds and/or trajectories which it has not trained on will be analyzed. The average tracking performance of the controller at various speeds will be determined, starting at low speeds which the controller has trained on and ending at high-speeds which it has not trained on.
5. **Validation & Comparison**
   The final sub-goal consists of validating the performance of the proposed adaptive DNN-based controller in terms of RMS position error and computational complexity. To do this a testing procedure will be designed that is repeatable and comparable to those used in other research papers. Additionally, if time allows other state-of-the-art controllers will be tested using identical testing procedures & drone to be able to accurately compare controller performance.

4

Methodologies

In the methodology section, first the problem statement is discussed in Section 4.1 followed by coordinate and notation definitions in Section 4.2. In Section 4.3, the quadrotor model as well as the aerodynamic model used are explained. Then quadrotor trajectory planning methods are discussed in Section 4.4 and finally high-speed trajectory tracking controllers are explained in detail in Section 4.5.

4.1. Problem Statement

Executing agile trajectories requires an accurate and efficient controller that tracks the desired trajectories. However, due the complex relationship between aerodynamic forces and quadrotor maneuverability at high speeds designing such a controller using traditional control design methods is insufficient.

Adaptive trajectory tracking of quadrotors is challenging due to complex and varying aerodynamic forces that cause significant disturbances and in turn introduce large positional errors. These aerodynamic effects are difficult to model on the scarce computational resources available on-board the quadrotor, yet are essential for accurate trajectory tracking of quadrotors. This justifies the design of a controller that is able to efficiently adapt to these aerodynamic effects in real-time versatile to propeller damage conditions, without a significant performance degradation.

4.2. Coordinate Frames

Two right-handed coordinate frames are used, shown in Figure 4.1. The inertial frame $\mathcal{F}^{\mathcal{I}} : \{\boldsymbol{x}^I, \boldsymbol{y}^I, \boldsymbol{z}^I\}$ with $\boldsymbol{z}^I$ pointing opposite direction of gravity and $\mathcal{F}^{\mathcal{B}} : \{\boldsymbol{x}^B, \boldsymbol{y}^B, \boldsymbol{z}^B\}$ with $\boldsymbol{z}^B$ pointing in the thrust direction and $\boldsymbol{x}^B$ pointing in the forward direction. Vectors expressed in the body frame are indicated by the superscript B while those expressed in the inertial frame have no superscript.

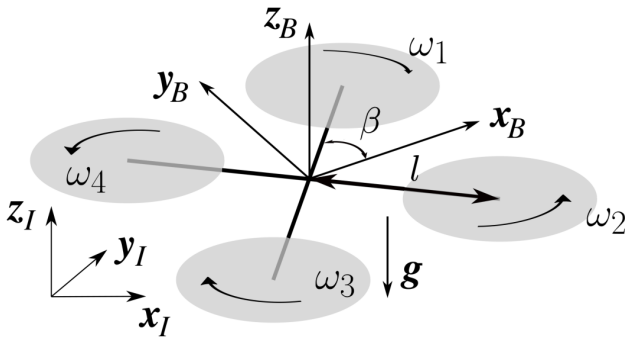


Figure 4.1: Coordinate Frame Definitions [36]

The rotation from $\mathcal{F}^{\mathcal{I}}$ to $\mathcal{F}^{\mathcal{B}}$ is represented by the rotational matrix $\boldsymbol{R}(\boldsymbol{q}) = \left[\boldsymbol{x}^B, \boldsymbol{y}^B, \boldsymbol{z}^B\right]$ parameterized by quaternion $\boldsymbol{q} = [q_\omega, q_x, q_y, q_z]^T$.

4.3. Quadrotor Model

The quadrotor is modelled as a 6 Degree of Freedom rigid body kinematic and dynamic model. The translational dynamics of a quadrotor are expressed in Equation 4.1 as:

$$\boldsymbol{a} = (\boldsymbol{T} + \boldsymbol{f}_a + \boldsymbol{f}_{res})/m + \boldsymbol{g}, \tag{4.1}$$

Where $\boldsymbol{a}$ is the position of the center of gravity of the quadrotor in the inertial frame; m is the mass of the quadrotor; $\boldsymbol{g}$ is the gravitational vector; $\boldsymbol{T}$ is the collective thrust modeled in Equation 4.4; $\boldsymbol{f}_a$ are the aerodynamic forces modeled using Equation 4.9 and $\boldsymbol{f}_{res}$ are the residual unmodeled forces not captured by the nominal model acting on the quadrotor.

The rotational kinematic and dynamic equations of a quadrotor are expressed in Equation 4.2 & Equation 4.3, respectively.

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\Omega}^B \end{bmatrix} \tag{4.2}$$

$$\boldsymbol{I}\dot{\boldsymbol{\Omega}}^B = \boldsymbol{I}\boldsymbol{\alpha}^B = \boldsymbol{\tau}^B + \boldsymbol{\tau}^B_{res} - \boldsymbol{\Omega}^B \times \boldsymbol{I}\boldsymbol{\Omega}^B \tag{4.3}$$

Where $\boldsymbol{\Omega}^B$ is the projection on $\mathcal{F}_{\mathcal{B}}$ of the angular velocity of $\mathcal{F}_{\mathcal{B}}$ with respect to $\mathcal{F}_{\mathcal{I}}$ directly measurable on the onboard Inertial Measurement Unit (IMU), $\boldsymbol{\Omega^B} = [p, q, r]^T$. Its derivative, the angular acceleration is is denoted by $\boldsymbol{\alpha}^B$. $\boldsymbol{I}$ is the mass moment of inertia matrix of the quadrotor and $\boldsymbol{\tau}$ are the torques generated by the rotors modeled in Equation 4.4 and $\boldsymbol{\tau}_{res}$ are the residual body torques.

The collective thrust $\boldsymbol{T}^B$ and torque $\boldsymbol{\tau}^B$ in the body frame are modeled as functions of the rotor Rotation Per Minute (RPM) speeds $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3, \omega_4]^T$:

$$\begin{bmatrix} \boldsymbol{T}^B \\ \boldsymbol{\tau}^B \end{bmatrix} = \boldsymbol{G}_1\boldsymbol{u} + \boldsymbol{G}_2\dot{\boldsymbol{\omega}} + \boldsymbol{G}_3(\boldsymbol{\Omega}^B)\boldsymbol{\omega} \tag{4.4}$$

where:

$$\boldsymbol{u} = c_t \sum_{i=0}^{4} \omega_i^2 \tag{4.5}$$

$$\boldsymbol{G_1} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ l\sin(\beta) & -l\sin(\beta) & -l\sin(\beta) & l\sin(\beta) \\ -l\cos(\beta) & -l\cos(\beta) & l\cos(\beta) & l\cos(\beta) \\ c_q/c_t & -c_q/c_t & c_q/c_t & -c_q/c_t \end{bmatrix} \tag{4.6}$$

$$\boldsymbol{G_2} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ I_r & -I_r & I_r & -I_r \end{bmatrix} \tag{4.7}$$

$$\boldsymbol{G_3} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ I_r q & -I_r q & I_r q & -I_r q \\ -I_r p & I_r p & -I_r p & I_r p \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.8}$$

where c_t is the thrust coefficient and c_q is the torque coefficient provided in Table 4.1; β and l are geometric parameters of the quadrotor defined in Figure 4.1; I_r is the inertia of the rotor along the z axis. Torques caused by the angular acceleration of the rotor's and the gyroscopic effects corresponding to terms $\boldsymbol{G}_2$ & $\boldsymbol{G}_3$ respectively are negligible and typically omitted.

4.3.1. Aerodynamic Model

Quadrotors will experience significant aerodynamic forces during high-speed flight that lead to large tracking errors. The parametric aerodynamic drag model from [37] shown in Equation 4.9 has proven to be effective at approximating these aerodynamic forces.

$$\boldsymbol{f}_a^B = \sum_{i=0}^{4} \omega_i \begin{bmatrix} -k_x v_x^B \\ -k_y k_y^B \\ -k_z v_z^B \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ k_h \left({v_x^B}^2 + {v_y^B}^2 \right) \end{bmatrix} \tag{4.9}$$

Where $\boldsymbol{R}(\boldsymbol{q})\, \boldsymbol{v^I} = \boldsymbol{v^B} \left[v_x^B, v_y^B, v_{z^B} \right]$ are the velocity components in the body frame and k_x, k_y, k_z, k_h are coefficients identified from flight data. Thankfully these coefficients have already been identified for the Parrot Bebop 2 drone in [11] and are shown in Table 4.1.

Table 4.1: Aerodynamic model coefficients for the Parrot Bebop 2 drone from [11]

Coefficient	Value
k_x	4.43E-6
k_y	3.96E-6
k_z	1.14E-5
k_h	2.57E-2
c_t	1.79E-8
c_q	6.60E−12

4.4. Trajectory Planning

Two common approaches for planning quadrotor trajectories are continuous-time polynomials and discrete-time state space representations. Within continuous-time polynomial trajectories two popular approaches are bang-bang and minimum snap trajectories with the latter discussed in detail in Section 4.4.1. With regards to discrete-time state space trajectories the method used in [12] to generate truly time-optimal trajectories that fully exploit the actuation limits of the quadrotors has been outlined in Section 4.4.2. Ultimately, minimum snap trajectories are used in the preliminary experiment due to the ease of their implementation.

4.4.1. Minimum Snap

Quadrotors are differentially flat systems [10] that can be described based on the flat output of the position & yaw states. The evolution of these flat outputs can be represented as smooth differentiable polynomials on time, which can be obtained by minimizing snap (4^{th} derivative of time) while satisfying waypoint and continuity constraints. Trajectories through many waypoints can be efficiently generated by concatenating separate polynomials for each waypoint segment. The reasoning behind using minimum snap trajectories is to ”minimize and smooth the required body torques and therefore single motor thrust differences to track the trajectory, which are dependant on the snap” [12].

Given a list of $\boldsymbol{N}$ waypoints and thus $M = N - 1$ trajectories, the desired position of the quadrotor for a waypoint segment m is described by $\boldsymbol{p}_r(t) = [x_m(t), y_m(t), z_m(t)]$ with each axis represented by a polynomial. The minimum snap trajectory is obtained by solving the minimization problem in Equation 4.10 in Python using the Constrained Optimization BY Linear Approximation (COBYLA) solver.

$$\boldsymbol{p}_r^\star(t) = \underset{\boldsymbol{p}_r(t)}{\arg\min} \sum_{m=1}^{M} \int_0^{T_m} \left\| \ddddot{\boldsymbol{p}}_{r_m}(t) \right\| dt \tag{4.10}$$

Subject to the start & end zero-velocity constraints in Equation 4.11, the waypoint constraints in Equation 4.12 where $\boldsymbol{p}_{w_m}$ denotes the position of waypoint m and the continuity constraints in Equation 4.13.

Subject to:

$$\dot{\boldsymbol{p}}_{r_1}(0) = \mathbf{0}, \quad \dot{\boldsymbol{p}}_{r_M}(T_M) = \mathbf{0} \tag{4.11}$$

$$\boldsymbol{p}_{r_m}(0) = \boldsymbol{p}_{w_m}, \quad \boldsymbol{p}_{r_m}(T_m) = \boldsymbol{p}_{w_{m+1}} \tag{4.12}$$

$$\dot{\boldsymbol{p}}_{r_m}(T_m) = \dot{\boldsymbol{p}}_{r_{m+1}}(0), \quad \ddot{\boldsymbol{p}}_{r_m}(T_m) = \ddot{\boldsymbol{p}}_{r_{m+1}}(0) \tag{4.13}$$

According to the theory of Euler-Lagrangian mechanics, the principle of least action dictates that the solution of the minimization problem must satisfy Equation 4.14.

$$\frac{\partial L}{\partial \boldsymbol{p}_r} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{\boldsymbol{p}}_r} + \cdots + \frac{\mathrm{d}^4}{\mathrm{d}t^4}\frac{\partial L}{\partial \ddddot{\boldsymbol{p}}_r} = 0 \tag{4.14}$$

where $L = ||\dddot{\boldsymbol{p}}_r||$. Indicating that the reference trajectory $\boldsymbol{p}_r$ should be represented as a 7^{th} order polynomial given that $\boldsymbol{p}_r^{(8)} = 0$.

The time allocated to each waypoint segment T is manually chosen. To ensure that the speed of the reference trajectory does not vary significantly between segments pain-staking tuning is required. Instead, the total time of the reference trajectory is included in the cost function as shown in Equation 4.15, where γ is the time penalty. However, given that the cost function and constraints are dependant on the segment times the minimization problem must now be solved iteratively starting with an initial guess of segment times and then using gradient descent.

$$\boldsymbol{p}_r^{\star}(t) = \underset{\boldsymbol{p}_r(t)}{\arg\min} \sum_{m=1}^{M} \left(\int_0^{T_m} \left\| \dddot{\boldsymbol{p}}_{r_m}(t) \right\| dt + \gamma T_m \right) \tag{4.15}$$

When penalizing only the snap of a trajectory "the cost function could be arbitrarily driven close to zero by increasing the total time, but Equation 4.15 has a definite optimal solution that only varies with γ" [30]. The effect of the time penalty γ on the minimum snap trajectory is shown in Figure 4.2.

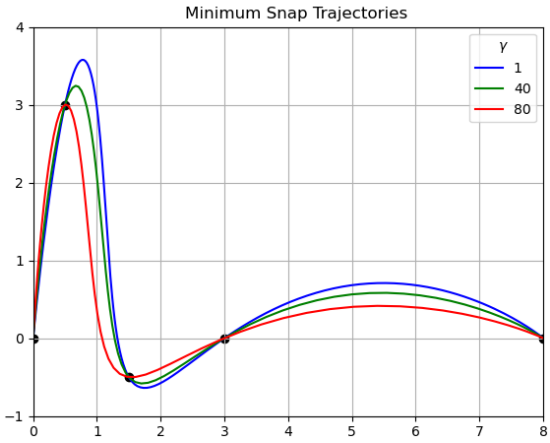


Figure 4.2: Effect of time factor γ on minimum snap trajectory. Total times of $T_M = [26.5s, 13.7s, 11.3s]$ with $\gamma = [1, 40, 80]$

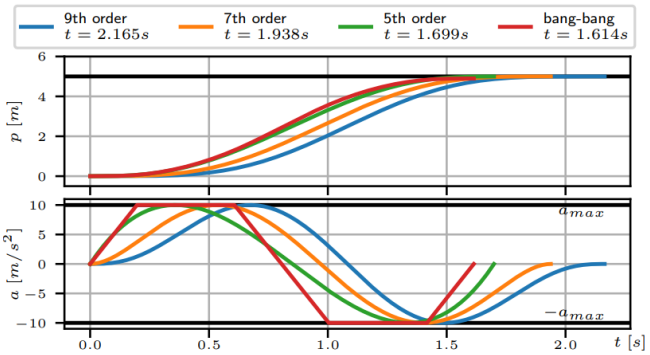


Figure 4.3: Position and acceleration of varying order polynomial and bang-bang trajectories. Bang-bang trajectory is shortest by spending more time at the acceleration limits. [12]

Minimum snap trajectories are by definition smooth which are relatively easy for a quadrotor to track but do not fully maximize the acceleration of the quadrotor at all times. Thus minimum snap trajectories are not truly agile high-acceleration trajectories. In fact, "due to the polynomial nature of minimum snap trajectories, the boundaries of the reachable input spaces can only be touched at one or multiple points, but not at sub-segments of the trajectory" [12] as can be seen by comparing minimum snap trajectories with a bang-bang trajectory in Figure 4.3.

4.4.2. Time-Optimal

Conventional discrete-time state space formulations are ineffective at time-optimal trajectory generation. If multiple waypoints must be passed through the trajectory, constraints must be allocated to specific nodes on the trajectory, However the time spent between waypoints is unknown.

This problem is solved in [12] by "introducing a formulation of progress along the trajectory allowing for the simultaneous optimization of the time-allocation and the trajectory itself" [12]. More specifically the proposed Complementary Progress Constraint (CPC) is composed of two complementary factors: the completion of a waypoint (progress) and the local proximity to a waypoint. This method allows for the generation of truly time-optimal trajectories that fully exploit the quadrotor's actuation limits.

The time-optimal trajectory is obtained by solving the optimization problem in Equation 4.16. Where x is the full space of optimization variables consisting of $\boldsymbol{x} = [t_N, \boldsymbol{x}_0, \ldots, \boldsymbol{x}_N]$, the overall time t_N and all variables assigned to nodes k as $x_k = [\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \boldsymbol{\nu}_k]$.

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} t_N \tag{4.16}$$

Subject to the system dynamics & initial constraint in Equation 4.17, the input constraints in Equation 4.18. Additionally subject to the progress evolution, boundary & sequence constraints in Equation 4.19 where $\boldsymbol{\lambda}$ is a vector defining the progress variables and the vector $\boldsymbol{\mu}$ indicates the progress change at every timestep. Finally, subject to the complementary progress constraint with tolerance in Equation 4.20 where ν_k^j is a slack variable allowing the distance to waypoint to be relaxed to zero when smaller than d_{tol}.

Subject to:

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k - dt \cdot \boldsymbol{f}_{RK4}(\boldsymbol{x}_k, \boldsymbol{u}_k) = 0 \tag{4.17}$$

$$\boldsymbol{x}_0 = \boldsymbol{x}_{init}$$

$$\boldsymbol{u}_{min} - \boldsymbol{u}_k \geq 0, \quad \boldsymbol{u}_k - \boldsymbol{u}_{max} \geq 0 \tag{4.18}$$

$$\boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_k + \boldsymbol{\mu}_k = \boldsymbol{0} \tag{4.19}$$

$$\boldsymbol{\lambda}_0 - 1 = \boldsymbol{0}, \quad \boldsymbol{\lambda}_N = \boldsymbol{0}, \quad \boldsymbol{\mu}_k \geq 0, \quad \boldsymbol{\lambda}_k^j - \boldsymbol{\lambda}_k^{j+1} \leq 0$$

$$\mu_k^j \cdot \left(\left\| \boldsymbol{p}_{r_k} - \boldsymbol{p}_{w_j} \right\|^2 - \nu_k^j \right) = 0 \tag{4.20}$$

$$-\nu_k^j \leq 0, \quad \nu_k^j - d_{\text{tol}}^2 \leq 0$$

In [12] this method is used to generate a time-optimal trajectory through 2.5 laps of racing track consisting of 18 waypoints with a maximum thrust-to-weight ratio of $T/W = 3.3$ in $t \sim 40\ [min]$ on a desktop computer. The generated trajectory is demonstrated to outperform professional human pilots in terms of lap times ($t_{hum} = 6.79\ [s]$ vs. $t_{opt} = 6.12\ [s]$) while staying within a safe margin of the quadrotors absolute limits ($T/W \sim 4$).

4.5. Controllers

Conventional high-speed trajectory tracking controllers can be classified into either predictive or non-predictive frameworks. Predictive methods encode multiple future time-steps into the control command while non-predictive methods only track a single reference step. Two state-of-the-art controllers are the Nonlinear Model Predictive Controller (NMPC) and the non-predictive Differential-Flatness-Based-Controller
(DFBC) both of which are explained further in Section 4.6 and Section 4.7 respectively.

The Neural-Fly controller architecture is outlined in Section 4.7.2, more specifically it's two main components: the offline learning phase and the online adaptation phase are explained in detail in Section 4.7.3 & Section 4.7.4 respectively. Neural-Fly is a standalone controller, however in practice it is an aerodynamic model at its core, which could be used to replace the aerodynamic model used in the NMPC or DFBC controllers to further improve performance.

4.6. Nonlinear Model Predictive Controller

Model predictive control is a ”popular method in robotics due to its predictive nature and ability to handle input constraints” [36]. The main principle of a model predictive controller (MPC) is to generate control commands that minimize the tracking error by solving an Optimal Control Problem (OCP) over a **receding horizon**.

Receding Horizon

The simplest method to obtain a control sequence that minimizes the trajectory tracking error is to use fixed horizon optimization. Fixed horizon optimization finds the optimal control sequence over a fixed interval $[u_i, ..., u_{i+N-1}$ starting at the current time step i and ending at some future time-step $i+N-1$.

However, the fixed horizon principle suffers from two major drawbacks:

- If there is an unexpected change in the system at some time over the predicted control sequence $[i, i+N-1]$ that is not predicted by the model it renders the predicted control sequence $[u_i, ..., u_{i+N-1}$ obsolete.
- As the control sequence approaches the final time-step $i+N-1$ the performance of the control law at tracking the desired trajectory decreases due to a lack in the reduction of the objective function.

The receding horizon optimization addresses these issues by determining the optimal control sequence over a fixed future interval at each time-step and then applying the first control step of this sequence. At time i with state x_i the optimal control sequence $[u_i, ..., u_{i+N-1}$ is determined, the first control step u_i is then applied. The optimal control sequence at time-step $i+1$ with the new state x_{i+1} is determined $[u_{i+1}, ..., u_{i+N}$, repeating the same process.

If disturbances are negligible or time-steps are short, the state of the system measured at the next time-step $i+1$ should be the same as the one predicted by the model. However using the measured state rather than the predicted state results in a more robust and accurate controller at the cost of a higher computational load.

First, the reference states $\boldsymbol{x}_r$ and inputs $\boldsymbol{u}_r$ are first obtained from a trajectory planner that can generate full states such as the one from [12]. The NMPC then minimizes the error between the predicted states and the reference states over a receding horizon discretized into N equal intervals, yielding the constrained nonlinear optimization problem of Equation 4.21.

$$\boldsymbol{u}_{\text{NMPC}} = \underset{\boldsymbol{u}}{\text{argmin}} \sum_{k=0}^{N-1} \left(\|\boldsymbol{x}_k - \boldsymbol{x}_{k,r}\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_k - \boldsymbol{u}_{k,r}\|_{\boldsymbol{Q}_u} \right) + \|\boldsymbol{x}_N - \boldsymbol{x}_{N,r}\|_{Q_N} \tag{4.21}$$

$$\begin{aligned} \text{s.t.} \qquad & \boldsymbol{x}_0 = \boldsymbol{x}_{init}, \\ & \boldsymbol{x}_{k+1} = \boldsymbol{f}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right), \end{aligned}$$

Where $\boldsymbol{u} \in [\boldsymbol{u}_{\text{min}} \ \ \boldsymbol{u}_{\text{max}}]$ is the vector of motor thrust defined in Equation 4.5 bounded to the range between the minimum and maximum thrust a motor can provide. The state vector x is defined as $\boldsymbol{x} = \left[\boldsymbol{p}\ \dot{\boldsymbol{p}}\ \boldsymbol{q}\ \boldsymbol{\Omega}^B\right]$ and the body angular rates are bounded to the range $\boldsymbol{\Omega}^B \in \left[\boldsymbol{\Omega}^B_{\text{min}} \ \ \boldsymbol{\Omega}^B_{\text{max}}\right]$ for improved stability.

MPC is very computationally demanding compared to non-predictive methods, especially when using Nonlinear-MPC with a full-state nonlinear model of a quadrotor. Thanks to advancements in small embedded computers & nonlinear solvers, running NMPCs with fully nonlinear dynamic models has recently become computationally tractable on quadrotors.

4.7. Differential-Flatness Based Controller

Quadrotors are differentially flat systems [10] meaning that "all states and inputs can be written as algebraic functions of the flat outputs and its derivatives" [10], allowing for the direct mapping of the flat outputs (position $\boldsymbol{p}$ and heading ψ) to the angular rates/accelerations. This property is leveraged by DFBC, including the position and heading outputs as feed-forward terms, improving the tracking accuracy.

First the desired acceleration is computed using the PD controller in Equation 4.22.

$$\boldsymbol{a}_d = \boldsymbol{K}_p \left(\boldsymbol{p}_r - \boldsymbol{p}\right) + \boldsymbol{K}_v \left(\boldsymbol{v}_r - \boldsymbol{v}\right) + \boldsymbol{a}_r \tag{4.22}$$

where $\boldsymbol{K_p}$ and $\boldsymbol{K_I}$ are positive-definite gain matrices. The desired thrust is computed by solving the translational dynamic model from Equation 4.1 in the thrust direction $z_{B,d}$ shown in Equation 4.23.

$$\boldsymbol{T}_d z_{B,d} = m\left(\boldsymbol{a} - \boldsymbol{g}\right) - R(q)\boldsymbol{f}_a \tag{4.23}$$

Given the reference heading angle ψ_r, the desired attitude of the quadrotor is obtained using Equation 4.24, Equation 4.25 & Equation 4.26.

$$\boldsymbol{x}_{C,d} = \left[\cos\left(\psi_r\right),\ \sin\left(\psi_r\right),\ 0\right]^T \tag{4.24}$$

$$\boldsymbol{y}_{B,d} = \frac{\boldsymbol{z}_{B,d} \times \boldsymbol{x}_{C,d}}{\left\|\boldsymbol{z}_{B,d} \times \boldsymbol{x}_{C,d}\right\|} \tag{4.25}$$

$$\boldsymbol{R}\left(\boldsymbol{q}_d\right) = \left[\boldsymbol{x}_{B,d},\ \boldsymbol{y}_{B,d},\ \boldsymbol{z}_{B,d}\right] \tag{4.26}$$

Taking the derivative of the translation dynamic model from Equation 4.22 and assuming constant aerodynamic force $\boldsymbol{f_a}$ we have Equation 4.27.

$$m\dot{\boldsymbol{a}} = \dot{\boldsymbol{T}}\boldsymbol{z}_B + \boldsymbol{T}\boldsymbol{\Omega} \times \boldsymbol{z}_B \tag{4.27}$$

Substituting jerk $\dot{\boldsymbol{a}}$ with reference jerk $\dot{\boldsymbol{a}}_r$ in Equation 4.27 and rearranging for $\boldsymbol{h}_\Omega \triangleq \boldsymbol{\Omega} \times \boldsymbol{z}_B$ yields Equation 4.28.

$$\boldsymbol{h}_\Omega = \left(m\dot{\boldsymbol{a}}_r - \dot{\boldsymbol{T}}\boldsymbol{z}_B\right) / \boldsymbol{T} \tag{4.28}$$

Where the collective thrust derivative $\dot{T}$, which cannot be directly measured, is approximated by the reference jerk $\dot{T} \approx m\dot{\boldsymbol{a}}_r \cdot \boldsymbol{z}_B$. The reference angular velocity can then be obtained by using Equation 4.29.

$$\boldsymbol{\Omega}_r^B = \left[-\boldsymbol{h}_\Omega \cdot \boldsymbol{y}_B,\ \boldsymbol{h}_\Omega \cdot \boldsymbol{x}_B,\ \dot{\psi}_r \boldsymbol{z}_I \cdot \boldsymbol{z}_B\right]^T \tag{4.29}$$

Deriving Equation 4.27 further as well as replacing snap $\dot{\boldsymbol{a}}$ with the reference snap $\dot{\boldsymbol{a}}_r$ and rearranging for $\boldsymbol{h}_{\boldsymbol{\alpha}} \triangleq \dot{\boldsymbol{\Omega}} \times \boldsymbol{z}_B$ yields Equation 4.30.

$$\boldsymbol{h}_{\boldsymbol{\alpha}} = \frac{m}{\boldsymbol{T}}\dddot{\boldsymbol{p}}_r - \left(\boldsymbol{\Omega} \times \left(\boldsymbol{\Omega} \times \boldsymbol{z}_B\right) + \frac{2\dot{\boldsymbol{T}}}{\boldsymbol{T}}\boldsymbol{\Omega} \times \boldsymbol{z}_B + \frac{\ddot{\boldsymbol{T}}}{\boldsymbol{T}}\boldsymbol{z}_B\right) \tag{4.30}$$

Similarly, the double derivative of the collective thrust is approximated by the reference snap $\ddot{T} = m\dot{\boldsymbol{a}}_r \cdot \boldsymbol{z}_B + m\left(\boldsymbol{\Omega} \times \boldsymbol{z}_B\right) \cdot \dot{\boldsymbol{a}}_r$. The desired angular acceleration can then be obtained with Equation 4.31.

$$\boldsymbol{\alpha}_r^B = \left[-\boldsymbol{h}_\alpha \cdot \boldsymbol{y}_B,\ \boldsymbol{h}_\alpha \cdot \boldsymbol{x}_B,\ \ddot{\psi}_r \boldsymbol{z}_I \cdot \boldsymbol{z}_B\right]^T \tag{4.31}$$

4.7.1. Tilt Prioritized Control

Quadrotors suffer from poor heading responsiveness, with control effectiveness around one order of magnitude lower than for pitch and roll often leading to motor saturation. Fortunately, the thrust orientation of a quadrotor is independent of it's heading angle, thus tilt-prioritized control proposed in [4] improves trajectory tracking performance while preventing motor saturation. Tilt-prioritized control regulates the reduced-attitude (pitch & roll) error $\tilde{\boldsymbol{q}}_{e,\theta\&\phi}$ and yaw error $\tilde{\boldsymbol{q}}_{e,\psi}$ separately computed in Equation 4.33 & Equation 4.34 respectively.

$$[q_{e,w}, q_{e,x}, q_{e,y}, q_{e,z}]^T = \boldsymbol{q}_d \otimes \boldsymbol{q}^{-1} \tag{4.32}$$

$$\tilde{\boldsymbol{q}}_{e,\theta\&\phi} = \frac{1}{\sqrt{q_{e,w}^2 + q_{e,z}^2}} \begin{bmatrix} q_{e,w}q_{e,x} - q_{e,y}q_{e,z} \\ q_{e,w}q_{e,y} + q_{e,x}q_{e,z} \\ 0 \end{bmatrix} \tag{4.33}$$

$$\tilde{\boldsymbol{q}}_{e,\psi} = \frac{1}{\sqrt{q_{e,w}^2 + q_{e,z}^2}} \begin{bmatrix} 0 & 0 & q_{e,z} \end{bmatrix}^T \tag{4.34}$$

The desired angular accelerations can then be computed using the attitude control law in Equation 4.35.

$$\boldsymbol{\alpha}_d^B = k_{q,\theta\&\phi}\tilde{\boldsymbol{q}}_{e,\theta\&\phi} + k_{q,\psi}\,\mathrm{sgn}\left(q_{e,w}\right)\tilde{\boldsymbol{q}}_{e,\psi} + \boldsymbol{K}_{\boldsymbol{\Omega}}\left(\boldsymbol{\Omega}_r^B - \boldsymbol{\Omega}^B\right) + \boldsymbol{\alpha}_r^B \tag{4.35}$$

Where $k_{q,\theta\&\phi}$ and $k_{q,\psi}$ are positive gains of the reduced-attitude and yaw control respectively, a $k_{q,\theta\&\phi}$ and $k_{q,\psi} >> k_{q,\psi}$ is desired for improved trajectory tracking performance while preventing motor saturation.

Finally, the thrust commands of each motor are determined by solving for $\boldsymbol{u}$ in Equation 4.4 resulting the direct-inversion control allocation in Equation 4.36, while satisfying the minimum and maximum thrust constraints in Equation 4.37.

$$\boldsymbol{u} = \boldsymbol{G}_1^{-1} \begin{bmatrix} \boldsymbol{T}_d \\ \boldsymbol{I}\boldsymbol{\alpha}_d^B + \boldsymbol{\Omega}^B \times \boldsymbol{I}\boldsymbol{\Omega}^B \end{bmatrix} \tag{4.36}$$

$$\boldsymbol{u}_{\text{DFBC}} = \max\left(\boldsymbol{u}_{\text{min}}, \min\left(\boldsymbol{u}, \boldsymbol{u}_{\text{max}}\right)\right) \tag{4.37}$$

4.7.2. Neural-Fly

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. Allowing the controller to quickly adapt to changing wind conditions while maintaining a high tracking accuracy.

The Neural-Fly controller can be divided into two main phases: offline learning & online adaptation, which have been explained in greater detail in Section 4.7.3 & Section 4.7.4 respectively. The purpose of the offline learning phase is to learn a wind condition independent model from real-world data collected in various conditions. While the goal of the online adaptation phase is to "update the wind-dependent linear coefficients using a composite of the position tracking error term and the aerodynamic force prediction error term" [25]. The wind-effect force estimate is then obtained by combining the wind-dependent coefficients with the output of the pre-trained DNN.

4.7.3. Offline-Learning

Given a dataset, the goal of the offline meta-learning phase is to learn a representation $\phi(\boldsymbol{x})$ that for any condition w a latent variable $\boldsymbol{A}(w)$ exists that allows $\phi(\boldsymbol{x})\,\boldsymbol{A}(w)$ to approximate the unmodeled forces $f(\boldsymbol{x}, w)$. This optimal ϕ representation can be found by solving the optimization problem in Equation 4.38, note that the representation ϕ is shared across all conditions but the optimal weight $\boldsymbol{A}_k$ is specific to each condition.

$$\min_{\phi, \boldsymbol{A}_1, \cdots, \boldsymbol{A}_K} \sum_{k=1}^{K} \sum_{i=1}^{N_k} \left\| \boldsymbol{y}_k^{(i)} - \phi\left(\boldsymbol{x}_k^{(i)}\right) \boldsymbol{A}_k \right\| \tag{4.38}$$

When a quadrotor tracks a trajectory at varying speeds the trajectory that is actually flown will vary vastly depending on the speed, additionally the distributions of the in data $\boldsymbol{x}$ will also vary significantly for example due to the drone pitching more aggressively during higher speed flight. As a consequence of the inherent domain shift in $\boldsymbol{x}$ caused by the change in condition w the optimization problem in Equation 4.38 will struggle to find an optimal representation of ϕ.This may lead to the DNN ϕ over-fitting to the data instead of finding a condition invariant representation. In other words, "the DNN may learn the shift in distributions of x across the different conditions such that the variation in the unmodeled forces $f\left(\boldsymbol{x}, w_1\right), ..., f\left(\boldsymbol{x}, w_K\right)$ is reflected via the distribution of $\boldsymbol{x}$ instead of the condition $w_1, ...w_K$" [25].

To solve the domain shift problem an adversarial network with loss function shown in Equation 4.39 is used. The adversarial network consists of two main components: the DNN representation ϕ and a discriminator h that predicts the condition index from the output of ϕ. These two networks compete against each other in a zero-sum game which allows the adversarial network to learn a condition invariant representation ϕ in an unsupervised manner.

$$\max_{h} \min_{\phi, \boldsymbol{A}_1, \cdots, \boldsymbol{A}_K} \sum_{k=1}^{K} \sum_{i=1}^{N_k} \left(\left\| \boldsymbol{y}_k^{(i)} - \phi\left(\boldsymbol{x}_k^{(i)}\right) \boldsymbol{A}_k \right\| - \alpha \cdot \text{loss}\left(h\left(\phi\left(\boldsymbol{x}_k^{(i)}\right)\right),\ k\right)\right) \tag{4.39}$$

Where loss$(\cdot)$ is a classification loss function such as cross-entropy loss and $\alpha \geq 0$ is a hyperparameter that controls the degree of regularization. The state vector is defined as $\boldsymbol{x} = [\boldsymbol{v}\ \boldsymbol{q}\ \boldsymbol{\omega}]$ but could potentially include other variables such as the angular velocities $\boldsymbol{\Omega}^B$.

The Domain Adversarially Invariant Meta-Learning (DAIML) algorithm shown in algorithm 1 is a gradient based meta-learning algorithm but with a least squares adaptation step.

Algorithm 1: Domain Adversarially Invariant Meta-Learning [25]

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

1 **while** *not converged* **do**
2 Randomly sample $\boldsymbol{D}^k \in \boldsymbol{D}$
3 Randomly sample two disjoint batches B^a and B from D^k
4 Solve $a^*(\phi) = \arg\min_a \sum_{i \in B^a} \left\| y_k^{(i)} - \phi\left(x_k^{(i)}\right) A \right\|$
5 **if** $\|A^*\| > \gamma$ **then**
6 $A^* = \gamma \cdot \frac{A^*}{\|A^*\|}$
7 **end**
8 Train ϕ with loss: $\mathcal{L}_\phi = \sum_{i \in B} \left(\left\| y_k^{(i)} - \phi\left(x_k^{(i)}\right) A^* \right\| - \alpha \cdot \text{loss}\left(h\left(\phi\left(x_k^{(i)}\right)\right),\ k\right)\right)$
9 **if** *rand()* $\leq \eta$ **then**
10 Train h with loss: $\mathcal{L}_h = -\sum_{j=1}^{K} \delta_{kj} \log\left(h\left(\phi\left(x_k^{(i)}\right)\right)^\top e_j\right)$
11 **end**
12 **end**

The DAIML algorithm can be divided into three steps:

1. The adaptation step (Lines 1-7) solves the least squares problem on the adaptation set B^a to find the optimal latent variable $\boldsymbol{A}^*$.
2. The training step (Line 8) updates the parameters of the DNN ϕ on the training set B using SGD based on the optimal latent variable $\boldsymbol{A}^*$ from the adaptation step.

3. The regularization step (Line 9-12) updates the parameters of the DNN h on training set B using Stochastic Gradient Descent (SGD).

DAIML builds upon the baseline adversarial network architecture of Equation 4.39 by adding additional features. In the adaptation step the latent variable A^* is a function of ϕ, therefore in the training step SGD will also back-propagate through A^* ensuring that the latent variable used remains optimal. The robustness of the adaptive controller is improved by normalizing the latent variable (Line 6) by ensuring that $\|A^*\| \leq \gamma$. Additionally, "spectral normalization is used during training of ϕ to control the Lipschitz property and improve the generalizability of the neural network" [25]. At each iteration, ϕ is updated while keeping h fixed, then with a certain probability h is updated while ϕ is fixed to improve the convergence of the adversarial network.

4.7.4. Online Adaptation

For the online adaptation phase there are two common approaches. In the first one, the adaptation phase adapts the entire model online [21, 2], via gradient descent, this however incurs a large computational cost. It is infeasible to run such adaptation onboard at the required frequency of the control loop $\geqslant 100\ Hz$. Additionally such a method lacks robustness, allowing for unpredictable outputs to lead to close-loop instability. In the second approach the online adaptation only adapts a small segment of the model [25, 32], which significantly reduces the computational burden onboard and provides robustness guarantees.

In the offline-learning phase the latent variable $\boldsymbol{A}^*$ was determined by minimizing the least-squares force prediction error, while this is sufficient for training, in the online phase the ultimate goal is minimizing the position tracking error.

A standard PID controller is the simplest solution but they typically only include PI feedback on position error, D feedback on velocity error and gravity compensation which only leads to local exponential stability about a fixed point. While this is sufficient during hover and gentle manoeuvres it leads to large tracking errors during agile flight. A nonlinear controller such as the one shown in Equation 4.40 could be used, which includes velocity PI feedback where $\boldsymbol{s} = \boldsymbol{v} - \boldsymbol{v}_r$ and model feedforward terms to account for the known system dynamics. Allowing the controller to perform well at tracking agile high-acceleration trajectories in the presence of nonlinearities, however it suffers from being slow to react to changes in the unmodeled dynamics and disturbance forces through the integral term.

$$\boldsymbol{u}_{\text{NL}} = \underbrace{m\boldsymbol{a}_r + \boldsymbol{f}_a + \boldsymbol{g}}_{\text{nominal model feedforward}} \underbrace{- \boldsymbol{K}\boldsymbol{s} - \boldsymbol{K}_I \int \boldsymbol{s} dt}_{\text{PI feedback}} \tag{4.40}$$

A composite adaptation law has been designed which solves the aforementioned issues based on a Kalman filter estimator. The latent variable update $\dot{\hat{\boldsymbol{A}}}$ "depends on both the prediction error of the dynamics model as well as the tracking error, which allows the adaptation law to quickly identify and adapt to new conditions without requiring persistent excitation" [25].

The online adaptation law can be expressed as the control law in Equation 4.41, the adaptation law in Equation 4.42 and the covariance update in Equation 4.43.

$$\boldsymbol{u}_{\text{NF}} = \underbrace{m\boldsymbol{a}_r + \boldsymbol{f}_a + \boldsymbol{g}}_{\text{nominal model feedforward}} - \underbrace{\boldsymbol{K}\boldsymbol{s}}_{\text{feedback}} - \underbrace{\phi(\boldsymbol{x})\,\hat{\boldsymbol{A}}}_{\text{learning feedforward}} \tag{4.41}$$

$$\dot{\hat{\boldsymbol{A}}} = \underbrace{-\lambda\hat{\boldsymbol{a}}}_{\text{regularization}} \underbrace{- \boldsymbol{P}\phi(\boldsymbol{x})^\top \boldsymbol{R}^{-1}(\phi(\boldsymbol{x})\,\hat{\boldsymbol{A}} - \boldsymbol{y})}_{\text{force error}} + \underbrace{\phi(\boldsymbol{x})^\top \boldsymbol{s}}_{\text{tracking error}} \tag{4.42}$$

$$\dot{\boldsymbol{P}} = -2\lambda\boldsymbol{P} + \boldsymbol{Q} - \boldsymbol{P}\phi(\boldsymbol{x})^\top \boldsymbol{R}^{-1}\phi(\boldsymbol{x})\,\boldsymbol{P} \tag{4.43}$$

The adaptation law replaces the integral term in the nonlinear controller in Equation 4.40 with the learned force term $\hat{\boldsymbol{f}} = \phi(\boldsymbol{x})\,\hat{\boldsymbol{A}}$ for faster model miss-match feedback. Where $\boldsymbol{u}_{NF}$ is the control law,

$\dot{\hat{\boldsymbol{A}}}$ denotes the latent variable update and $\boldsymbol{P}$ is the estimate covariance matrix used for gain tuning, y is the residual force measurement $f(\boldsymbol{x}, w) = y + \epsilon$ with measurement noise ϵ. $\boldsymbol{Q}, \boldsymbol{R}$ are the process noise covariance matrix and the observation noise covariance matrix respectively while λ is the damping gain. Note that the controller takes velocity feedback despite the ultimate goal being minimizing the position tracking error as it "simplifies the analysis and gain tuning without losing accuracy" [25].

A Kalman-filter estimator is used to update the latent variable $\boldsymbol{A}$ as it is the optimal estimator that minimizes the variance of the parameter error [15]. Analyzing the Kalman-filter, the regularization term λ ensures that the latent variable update does not "blow-up" when there is a lack of persistent excitation of the learned model ϕ, useful when using deep multi-layered networks. The matrix $\boldsymbol{Q}$ tracks how quickly the environment/conditions change while $\boldsymbol{R}$ tracks the representation error $\boldsymbol{d}$ defined as $f(\boldsymbol{x}, w) = \phi(\boldsymbol{x})\hat{A} + \boldsymbol{d}$. Simply combining the Kalman-filter into the controller may lead to instabilities in the closed-loop system, thus the adaptation law also includes a tracking error term, making it a composite adaptation law, guaranteeing stability and simplifying gain tuning.

"The control & adaptation law have been designed such that the closed-loop stability is robust to imperfect learning and time-varying conditions" [25]. In terms of theoretical guarantees, the tracking error has been proven in [25] to exponentially converge to an error ball with size proportional to the representation error and measurement noise $||\boldsymbol{d} + \boldsymbol{\epsilon}||$ and $||\dot{\boldsymbol{A}}||$ the change in the latent variable.

5

Data Collection

In this chapter the experimental set-up is outlined in Section 5.1. In Section 5.2 the data collection as well as some data analysis is included. Lastly, the data processing is explained in Section 5.3.

5.1. Experimental Set-up

The experiments will be performed at the Micro Air Vehicle Laboratory (MAVLab) Department of the TU Delft Faculty of Aerospace Engineering [20]. The experimental-setup is divided into three categories: the platform (drone) in Section 5.1.1, the software in Section 5.1.2 and the environment in Section 5.1.3.

5.1.1. Platform

The Parrot Bebop 1 shown in Figure 5.1 is the chosen platform for the experimentation phase. The Bebop is a cheap commercial drone that can run custom autopilot software that has been used extensively at the MAVLab. It is the ideal platform for fast-prototyping as it has been thoroughly tested and is readily available, and for these reasons has been chosen for the initial experimentation phase. However, the bebop is an eight year old under-powered platform, having a thrust-to-weight ratio of $T/W \approx 1.5$ compared to $T/W \geq 4$ on modern racing drones such as the owned by the author shown in Figure 5.2. Given that the bebop can only reach speeds of up to $3\ [m/s]$ a more powerful drone may be used to reach higher speeds to properly design a high-speed trajectory tracking controller.

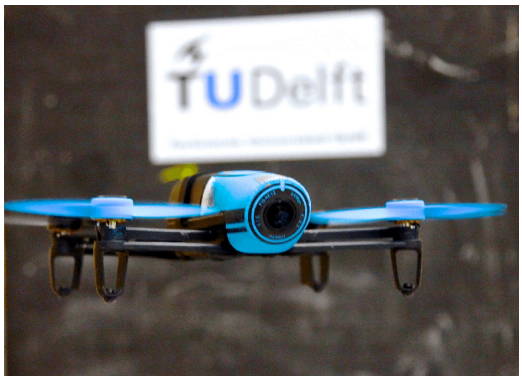


Figure 5.1: Picture of Parrot Bebop 1 flying in the CyberZoo [7]

Figure 5.2: Picture of a freestyle drone (Nazgul Evoque F5) [23]

5.1.2. Software

Customizability, good documentation, and hardware compatibility should all be considered when choosing an autopilot software. The Paparazzi-UAV open-source autopilot [27] is the chosen software as it is highly customizable and is already flashed on the Parrot Bebop 1, furthermore it has been used for

many years at the MAVLab, thus gathering extensive documentation and experience. Paparazzi-UAV is coded in C, which is slightly faster than C++ but also lacks some features and has fewer modules. There are other autopilots that also satisfy these requirements such as PX4 Autopilot [28] or Ardupilot [1], both of which are coded in C++ and are compatible with newer commercial flight controllers. All of these three autopilot software are suitable options to use on a more powerful drone, the final choice being subject to hardware compatibility between the autopilot software and the onboard flight controller.

The offline-learning phase will be performed in python. Python was the chosen programming language given the wide availability of easy-to-use & well documented machine learning modules such as Pytorch [29] or Keras [17] in addition to the considerable experience I have using these modules.

5.1.3. Environment

The environment where flights will be performed is the CyberZoo, shown in Figure 5.3. The CyberZoo is a research and test laboratory for flying and walking robots, it is a cage measuring $10 \times 10 \times 7\ m^3$ [35] located in the Delft Aerospace Structures and Materials Laboratory. The CyberZoo is equipped with an OptiTrack motion capture system, consisting of eight infra-red cameras, which provide accurate estimates of the drone's position and orientation indoors, ideal for data collection and validation. However, for higher-speed flights the CyberZoo is simply too small, in [36] they used a $30 \times 30 \times 8\ [m^3]$ flight volume motion capture system to reach speeds of up to $20\ [m/s]$. To achieve these high-speed conditions alternatives such as flying outdoors using GPS as shown in Figure 5.4 could be used.

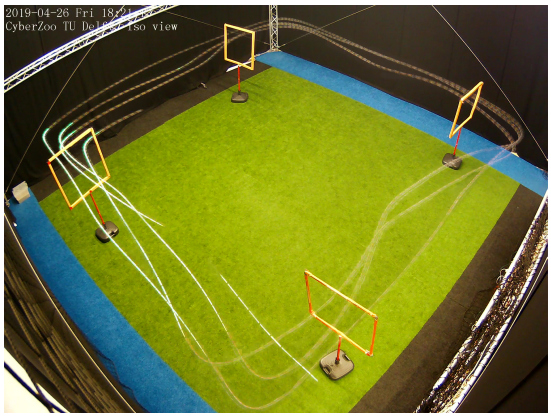


Figure 5.3: Long-exposure picture showing a drone flying through various gates in the CyberZoo [8]

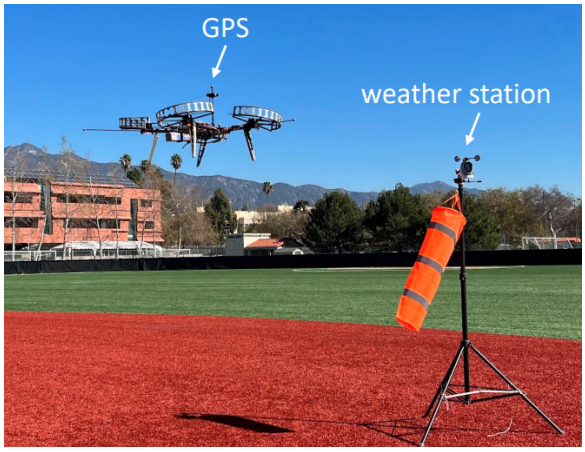


Figure 5.4: Picture of the outdoor experiments performed in [25] using a GPS module for state estimation and a weather station for wind data [25]

5.2. Data Collection

Data is required to learn a propeller damage invariant representation of the residual forces using the DAIML algorithm. Data collection consists of logging the relevant drone states: velocity, orientation and motor speeds while the drone tracks a randomized minimum snap trajectory from Section 4.4.1 using a baseline PID quaternion controller at varying propeller damage conditions.

For the purpose of the initial experimentation phase data collected for only one propeller damage condition: no propeller damage with $\alpha = 0$ corresponding to the non-adversarial training case. A reference and measured drone trajectory is shown in Figure 5.5, as can be seen there are significant tracking errors specially in the vertical direction.

The set of input-output pairs of the trajectory with speed factor w is referred to as sub-dataset D_w^k with $k = 1$ corresponding to the no propeller damage condition. The training dataset consists of six

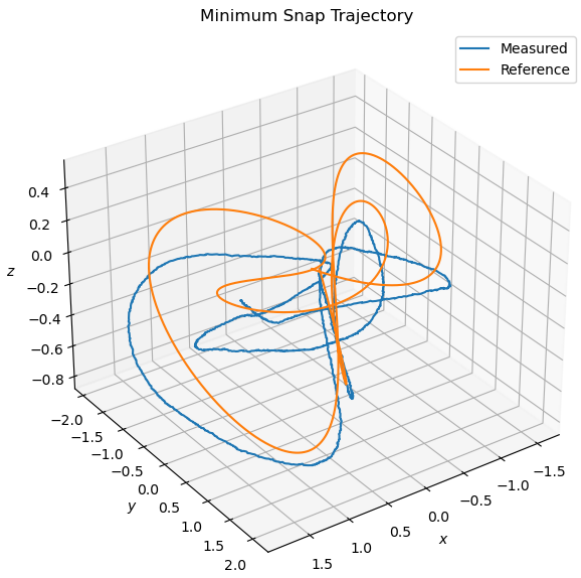


Figure 5.5: Sample reference vs. measured drone trajectory

different trajectories with speed factors $\omega = [0.2, 0.4, 0.6, 0.8, 1.1, 1.4]$ and reference velocities ranging from -0.3 to 3.0 $[m/s]$. The testing dataset consists of two trajectories with speed factors unseen in training $\omega = [0.7, 0.9]$. Information regarding the sub-dataset trajectories is included in Table 5.1.

Table 5.1: Total time, total distance and amount of data points of each reference trajectory recorded

Sub-Dataset (D_w^1)		**Speed Factor (w)**	**Total Time (T_{end}) [s]**	**Total Distance (d_{tot}) [m]**	**Datapoints**
Train	**1**	0.2	264.1	85.2	27,045
	2	0.4	123.8	76.7	12,676
	3	0.6	97.9	95.3	10,021
	4	0.8	68.5	85.6	7,011
	5	1.1	43.3	65.4	4,438
	6	1.4	43.8	87.7	4,481
Test	**1**	0.7	152.2	115.2	15,586
	2	0.9	76.9	81.8	7,875

The probability density functions of the reference velocity of training trajectories are shown in Figure 5.6. The speed factor has a significant influence on the reference velocity distribution. However, there is a significant overlap in reference velocities at low speeds, this in-balance in the training data should be addressed at a later stage to prevent the network from over-fitting to the most common velocity regime.

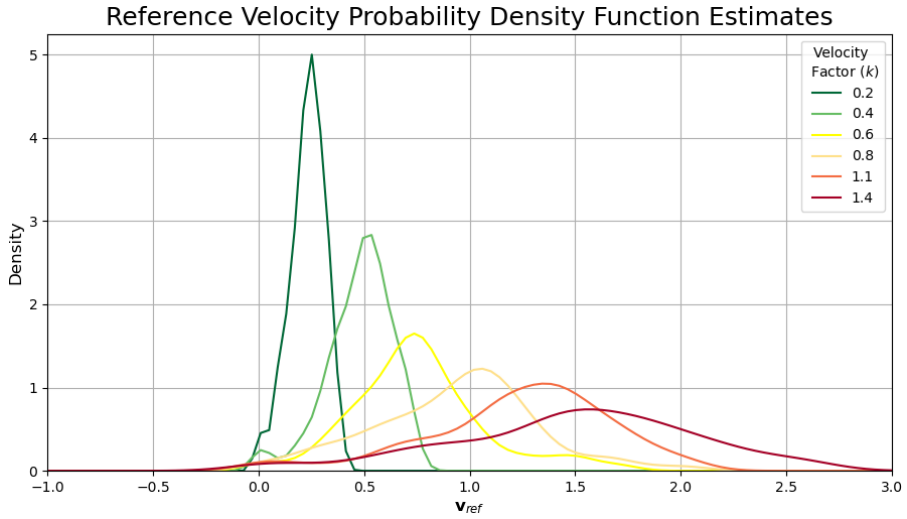


Figure 5.6: Training dataset probability density functions of reference velocity

5.3. Data Processing

The data collected is then processed to calculate the residual force acting on the drone. The measured forces are compared to the nominal quadrotor model comprised of the the thrust approximated in Section 4.3 and aerodynamic forces from Section 4.3.1.

First, the velocity measurements must be filtered. As can be seen in Figure 5.7 there is significant noise in the velocity measurements which would be further amplified when differentiating to obtain acceleration. Various smoothing methods for the velocity measurements, such as moving average, exponentially weighted moving average and low-pass filtering have been considered. Ultimately low pass zero-phase ”filtfilt” filtering is used as it removes most of the noise components without introducing lag by applying a filter forwards and then backwards. Allowing for the calculation of the residual force that is closest to the true ”ground-truth” value.

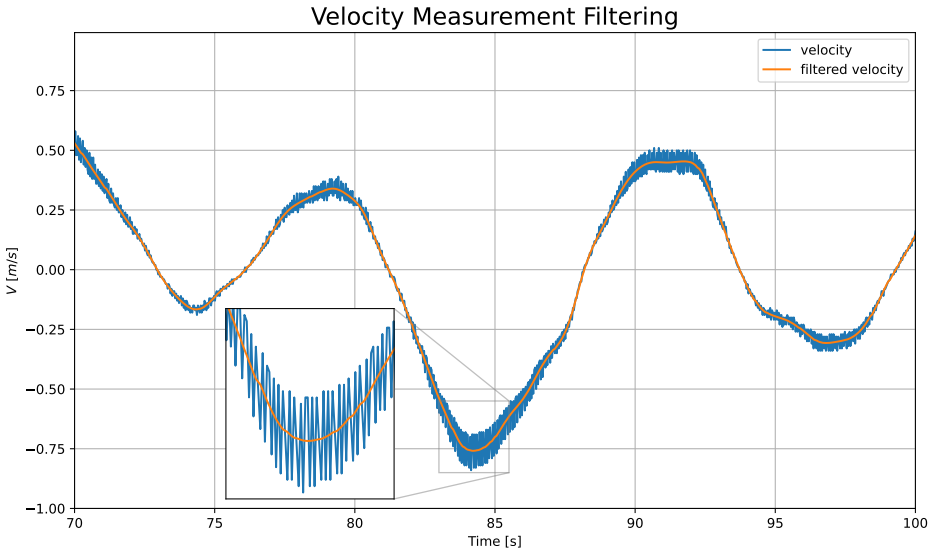


Figure 5.7: Smoothing of quadrotor x-direction velocity measurements using low-pass filtering with $\omega_c = 6/f_{\text{nyq}}$

Then the acceleration is obtain through first-order central finite difference of the velocity measurements given by Equation 5.1 with error of order h^4.

$$f_i' = \frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{12h} + O\left(h^4\right) \tag{5.1}$$

where h is the uniform grid spacing and the grid position is given by $f_{i+1} = f(x_i + h)$. Finally the residual force can calculated using the computed acceleration measurement by solving Equation 4.1 for $\boldsymbol{f_{res}}$ which yields Equation 5.2.

$$\boldsymbol{f}_{res} = m\left(\boldsymbol{a}_{meas.} - \boldsymbol{g}\right) - \boldsymbol{T} - \boldsymbol{f}_a \tag{5.2}$$

The thrust and aerodynamic model perform well at approximating the forces acting on the quadrotor, with only a slight offset in the z-component residual forces as seen in the residual force boxplots in Figure 5.8. The residual lateral forces vs speed are plotted in Figure 5.9.

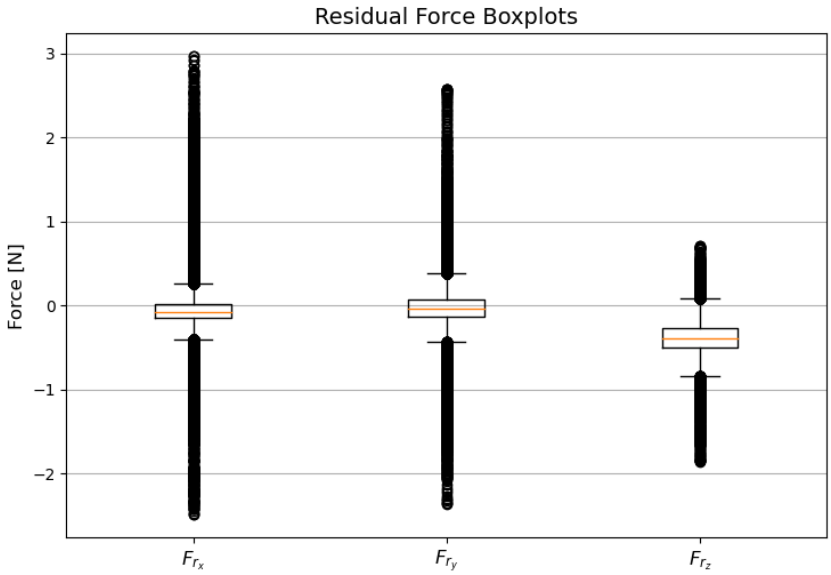


Figure 5.8: Boxplots of Residual Force

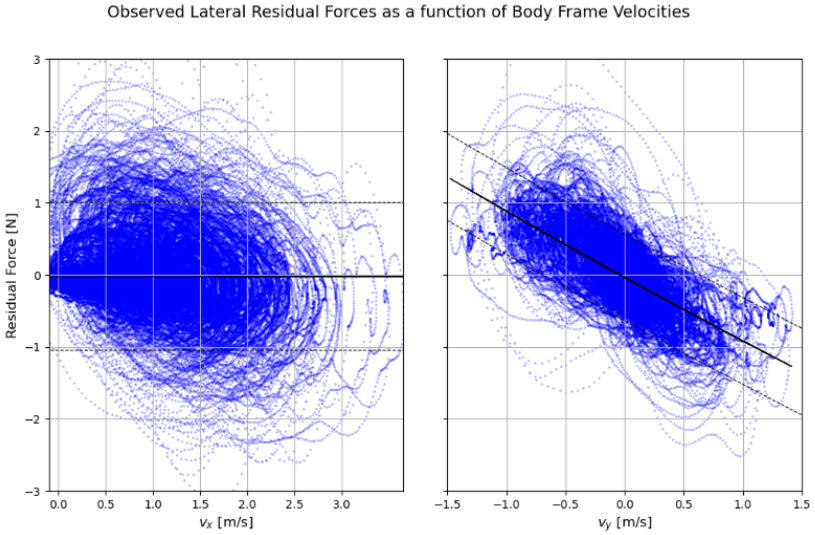


Figure 5.9: Lateral Residual Force vs Speed Plots

6 Training

The meta-learning DAIML algorithm is used to train a DNN on the data collected in Section 5.2 with the hyperparameters in Table 6.1. The DAIML algorithm has already been implemented in python with pytorch in [26], this implementation has been modified to allow for streamlined logging and hyperparameter sweeps using the wandb module, a sample log from a training run in the wandb website is shown in Figure 6.3 .

Table 6.1: DAIML Hyperparameters [25]

Hyperparameter	Value
Regularization (α)	0
Normalization (γ)	10
Spectral Normalization (SN)	2
Adaptation Batch Size (B^a)	128
Training Batch Size (B)	256
Learning Rate ϕ (lr_ϕ)	5E-4
Learning Rate h (lr_h)	1E-3
Architecture ϕ Net	$[11,\ 50,\ 60,\ 50,\ 4]$
Architecture h Net	$[4,\ 128,\ 6]$
Loss Type h	Cross-Entropy
Training Frequency h (γ)	0.5

Important DAIML hyperparameters are explained:

- The regularization parameter ($\alpha \geq 0$) determines the weight of the discriminator network h loss in the loss function in Equation 4.39. Having $\alpha > 0$ avoids over-fitting and ensures that the learnt ϕ network is to an extent invariant to speed conditions, however if $\alpha >> 0$ it may degrade the residual force prediction performance of ϕ. Note that given only one propeller damage condition has been recorded for this initial experimentation phase the regularization parameter used is $\alpha = 0$ corresponds to non-adversarial learning.
- Spectral normalization is a technique used for GANs to stabilize the training of the discriminator h. It normalizes the spectral norm of the weight matrix $\boldsymbol{W}$ of each layer of the discriminator such that it satisfies the Lipschitz constraint $\sigma(\boldsymbol{W}) = 1$.
- The discriminator training frequency $0 < \gamma \leq 1$ controls how often the discriminator network is trained each time the ϕ network is trained. A value of $\gamma = 1$ corresponds to training h and ϕ at each iteration while $\gamma = 0.1$ corresponds to training h once for every 10 times ϕ is trained. "A value of $\gamma = 0.5$ is commonly used for training stability for GANs" [25].

The training procedure of the meta-learning DAIML algorithm is outlined in algorithm 1 and illustrated in Figure 2.5 but is explained in further detail as follows. For each epoch the sub-dataset for

each propeller condition is chosen in a random order. Then, for each sub-dataset a training & adaptation batch are randomly sampled. The output of ϕ network on the adaptation batch is determined, then the optimal linear coefficients A^* are determined by performing ordinary least squares. Afterwards, the ϕ network and discriminator h network are trained normally on the training batch using the linear coefficients A^* determined previously.

The training & validation loss using the DAIML algorithm trained on the data collected in Section 5.2 are shown in Figure 6.1 & Figure 6.2 respectively. The training loss is composed of the force prediction loss which reflects how well ϕ approximates the residual forces and the classification loss which represents how well the discriminator h classifies the output of ϕ into a propeller damage condition. However, given that data for only one condition is recorded the classification loss is zero for this case. The validation loss is entirely composed of the force prediction loss.

Figure 6.1: ϕ Training Loss

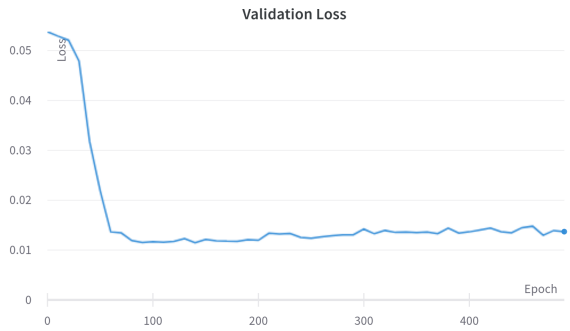


Figure 6.2: ϕ Validation Loss

As can be seen in training loss, the DAIML algorithm quickly converges to a quasi-minimum within 100 epochs indicating that the DAIML algorithm quickly learns the residual forces acting on the quadrotor. However, past 100 epochs the DAIML algorithm struggles to minimize the loss further indicating that DAIML is over-fitting the training data, this is also supported by the fact the validation loss does not decrease further past 100 epochs. To avoid over-fitting more training data should be used or if over-fitting persists the DAIML algorithm saved at 100 epochs should be used in the online adaptation phase.

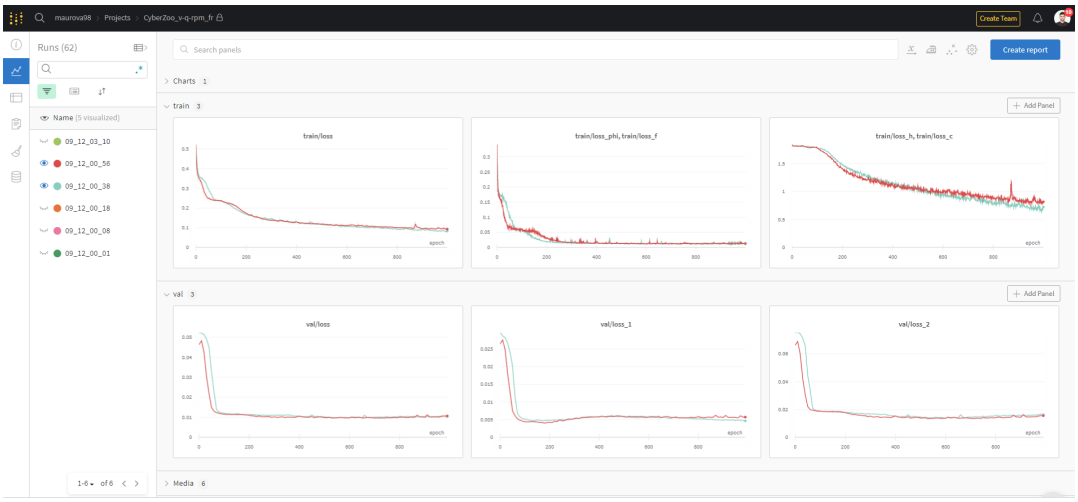


Figure 6.3: Sample log of DAIML training run in wandb

7

Thesis Planning

A project plan has been proposed by dividing the tasks mentioned in Chapter 3 into 23 work packages. These work packages have been incorporated into a work schedule that spans a duration of 8 months and consists of 3 phases: literature, mid-term and final review culminating in milestones shown as a Gantt Chart in Figure 7.1. Out of these work packages it is important to note that the ordering/building of the high-speed drone must occur before the validation in the final review phase.

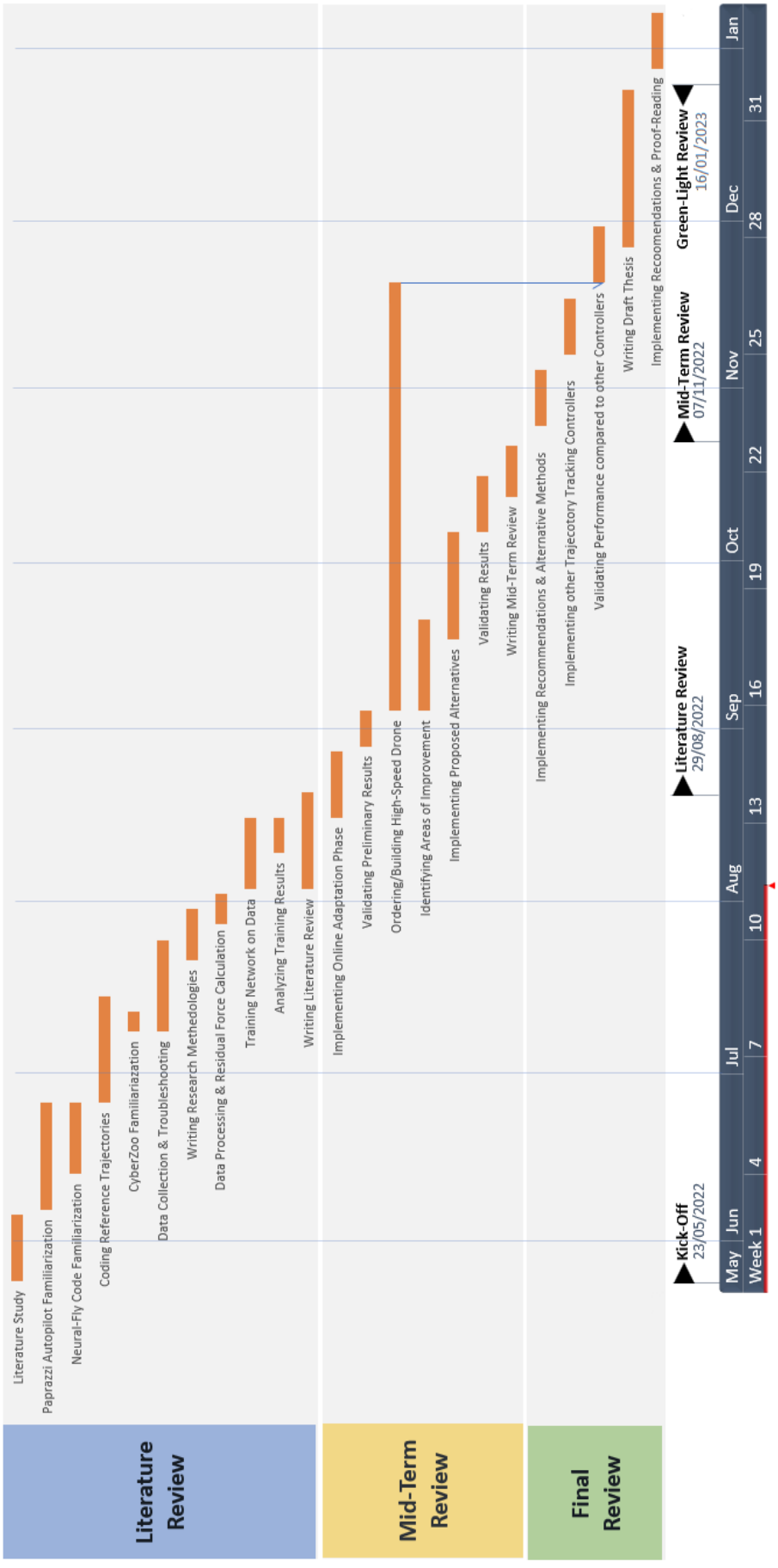


Figure 7.1: Gantt Chart

8

Results, Discussion & Relevance

The DAIML algorithm has shown promising results, quickly learning the residual forces from our custom dataset. However many unknowns remain, namely regarding the implementation and performance of the online-adaptation phase as well as the effect of propeller damage on the aerodynamic model. Furthermore, the dataset generation should ideally be comprised of uniform distribution of velocities to prevent the NN from over-fitting to a certain velocity regime.

The proposed controller will be validating by testing the controller on unseen minimum snap trajectories and measuring the RMS position error between the reference and measured trajectory of the drone. The onboard computational time to run the proposed controller will also be measured to asses the computational efficiency of the adaptive neural controller implementation. The performance of the proposed controller will then be compared against other trajectory tracking controllers in the same or similar conditions to obtain a good picture of the state of field. Verification steps will be performed by comparing the results obtained with those shown in [25] and to the data & code they have released.

The relevance of my thesis is to research the viability of adaptive DNN based controllers for trajectory tracking with varying propeller damage conditions. The performance of the adaptive neural controller is compared to state-of-the-art controllers in terms of accuracy and computational time. This research could motivate others to pursue further research in this field, aiding in the deployment of accurate trajectory tracking controllers for UAVs, allowing UAVs to fly through cluttered environments with a wide range of operating conditions.

9

Conclusion

An accurate & light-weight adaptive trajectory tracking controller is necessary to safely fly quadrotors with varying conditions in cluttered environments, such as in search & rescue, aerial delivery/transport, space exploration and autonomous drone racing. However, developing such a controller is a challenging task due to the complex aerodynamic disturbances that introduce large tracking errors during flight.

State-of-the-art adaptive controllers achieve impressive accuracies of $8.2\ cm$ at speeds of up to $20m\ s^{-1}$ [36] with slight degradation in performance to varying condition. However, they require powerful embedded computers or expensive sensors to reach these accuracies which hinders their wider adaptation in commercial applications. Recent work on adaptive Deep Neural Network based controllers [25] has shown promising results as a computationally light-weight alternative to these state-of-the-art controllers. DNN based controllers leverage the representation power of DNN to quickly learn the complex disturbances acting on the quadrotor which combined with a simple yet effective online adaptation phase allows for the controller to accurately track trajectories with a low computational burden even in unseen conditions. Despite these promising results there is a lack of research on designing such a controller for different conditions, with former experimentation being limited to tracking simple circular of figure-8 trajectories in a wind tunnel.

The aim of this thesis is to investigate an adaptive DNN based controller for trajectory tracking with propeller damage. In this literature review the groundwork to achieve this goal has been laid. First, a literature review on the progress and state-of-the-art of trajectory tracking controllers has been included in Chapter 2. Followed by an explanation of relevant research questions and objectives regarding developing such a controller in Chapter 3. In Chapter 4 the methodologies regarding the trajectory planning and controller have been outlined. The data collection & processing procedure have been explained in Chapter 5 while the training results of the DAIML algorithm have been analyzed in Chapter 6. Lastly, the results, outcome & relevance of this thesis have been discussed in Chapter 8.

References

[1] *ArduPilot*. URL: https://ardupilot.org (visited on 07/30/2022).

[2] S. Belkhale et al. “Model-Based Meta-Reinforcement Learning for Flight with Suspended Payloads”. In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 1471–1478. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2021.3057046. URL: http://arxiv.org/abs/2004.11345 (visited on 07/28/2022).

[3] M. Bisheban and T. Lee. *Geometric Adaptive Control with Neural Networks for a Quadrotor UAV in Wind fields*. arXiv:1903.02091 [math]. Mar. 2019. URL: http://arxiv.org/abs/1903.02091 (visited on 05/23/2023).

[4] D. Brescianini and R. D’Andrea. “Tilt-Prioritized Quadrocopter Attitude Control”. In: *IEEE Transactions on Control Systems Technology* 28.2 (Mar. 2020). Conference Name: IEEE Transactions on Control Systems Technology, pp. 376–387. ISSN: 1558-0865. DOI: 10.1109/TCST.2018.2873224.

[5] F.-C. Chen and H. Khalil. “Adaptive control of a class of nonlinear discrete-time systems using neural networks”. In: *IEEE Transactions on Automatic Control*. Vol. 40. May 1995, pp. 791–801. DOI: 10.1109/9.384214.

[6] Fu-Chuang Chen and H. Khalil. “Adaptive control of a class of nonlinear discrete-time systems using neural networks”. en. In: *IEEE Transactions on Automatic Control* 40.5 (May 1995), pp. 791–801. ISSN: 00189286. DOI: 10.1109/9.384214. URL: http://ieeexplore.ieee.org/document/384214/ (visited on 05/23/2023).

[7] G. de Croon. *TOP grant on self-supervised learning*. Section: Geen categorie. May 2018. URL: http://www.bene-guido.eu/wordpress/2018/05/16/top-grant-on-self-supervised-learning/ (visited on 07/30/2022).

[8] *Delftse wetenschappers maken kleinste autonome race-drone ter wereld*. URL: https://www.tudelft.nl/2019/tu-delft/delftse-wetenschappers-maken-kleinste-autonome-race-drone-ter-wereld (visited on 07/30/2022).

[9] *Drone Market in 2021-2026 Infographic | Drone Industry Insights*. Diagram. URL: https://droneii.com/project/drone-market-in-2021-2026 (visited on 07/26/2022).

[10] M. Faessler, A. Franchi, and D. Scaramuzza. “Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories”. In: *Robotics and Automation Letters (RA-L)*. Vol. 3. 2. IEEE Robotics and Automation Letters, Apr. 2018, pp. 620–626. DOI: 10.1109/LRA.2017.2776353. URL: http://arxiv.org/abs/1712.02402 (visited on 05/23/2022).

[11] R. Ferede. “An Adaptive Control Strategy for Neural Network based Optimal Quadcopter Controllers”. PhD thesis. 2022. URL: https://repository.tudelft.nl/islandora/object/uuid%5C%3Ab43a9703-082c-47c7-a56e-d50794ee8c1c (visited on 07/31/2022).

[12] P. Foehn, A. Romero, and D. Scaramuzza. “Time-Optimal Planning for Quadrotor Waypoint Flight”. In: *Science Robotics*. Vol. 6. 56. American Association for the Advancement of Science, July 2021, eabh1221. DOI: 10.1126/scirobotics.abh1221. URL: http://arxiv.org/abs/2108.04537 (visited on 05/23/2022).

[13] E. Frazzoli, M. Dahleh, and E. Feron. “Trajectory tracking control design for autonomous helicopters using a backstepping algorithm”. In: vol. 6. Feb. 2000, pp. 4102–4107. DOI: 10.1109/ACC.2000.876993.

[14] D. Hanover et al. “Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors”. In: *IEEE Robotics and Automation Letters*. Vol. 7. 2. IEEE, Apr. 2022, pp. 690–697. DOI: 10.1109/LRA.2021.3131690. URL: http://arxiv.org/abs/2109.04210 (visited on 05/23/2022).

[15] R. E. Kalman and R. S. Bucy. “New Results in Linear Filtering and Prediction Theory”. In: *Journal of Basic Engineering* 83.1 (Mar. 1961), pp. 95–108. ISSN: 0021-9223. DOI: 10.1115/1.3658902. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.361.6851&rep=rep1&type=pdf (visited on 08/11/2022).

[16] E. Kaufmann et al. “Deep Drone Acrobatics”. In: *Robotics, Science, and Systems (RSS)*. IEEE, June 2020. DOI: 10.48550/arXiv.2006.05768. URL: http://arxiv.org/abs/2006.05768 (visited on 05/23/2022).

[17] *Keras: the Python deep learning API*. URL: https://keras.io/ (visited on 07/30/2022).

[18] S. Khatoon, D. Gupta, and L. K. Das. “PID & LQR control for a quadrotor: Modeling and simulation”. In: *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. Sept. 2014, pp. 796–802. DOI: 10.1109/ICACCI.2014.6968232.

[19] P. Kotaru, R. Edmonson, and K. Sreenath. *Geometric L1 Adaptive Attitude Control for a Quadrotor Unmanned Aerial Vehicle*. arXiv:1910.07730 [math]. Mar. 2020. DOI: 10.48550/arXiv.1910.07730. URL: http://arxiv.org/abs/1910.07730 (visited on 05/21/2023).

[20] *MAVLab*. URL: https://mavlab.tudelft.nl/ (visited on 07/30/2022).

[21] A. Nagabandi et al. *Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning*. Feb. 2019. DOI: 10.48550/ARXIV.1803.11347. URL: http://arxiv.org/abs/1803.11347 (visited on 07/28/2022).

[22] K. Narendra and S. Mukhopadhyay. “Adaptive control using neural networks and approximate models”. In: *IEEE Transactions on Neural Networks*. Vol. 8. May 1997, pp. 475–485. DOI: 10.1109/72.572089.

[23] *Nazgul Evoque F5 4S/6S Analog FPV Drone- BNF*. URL: https://shop.iflight-rc.com/Nazgul-Evoque-F5-Analog-BNF-Pro1630 (visited on 07/30/2022).

[24] K. Northon. *NASA’s Ingenuity Mars Helicopter Succeeds in Historic First Flight*. Website. Apr. 2021. URL: http://www.nasa.gov/press-release/nasa-s-ingenuity-mars-helicopter-succeeds-in-historic-first-flight (visited on 07/26/2022).

[25] M. O’Connell et al. “Neural-Fly Enables Rapid Learning for Agile Flight in Strong Winds”. In: *Science Robotics*. Vol. 7. 66. American Association for the Advancement of Science, May 2022, eabm6597. DOI: 10.1126/scirobotics.abm6597. URL: http://arxiv.org/abs/2205.06908 (visited on 05/23/2022).

[26] M. O’Connell et al. *Public facing code for Neural Fly*. en. Apr. 2022. URL: https://github.com/aerorobotics/neural-fly (visited on 09/11/2022).

[27] *PaparazziUAV*. URL: https://wiki.paparazziuav.org/wiki/Main_Page (visited on 07/30/2022).

[28] *PX4 Autopilot*. URL: https://px4.io/ (visited on 07/30/2022).

[29] *PyTorch*. URL: https://www.pytorch.org (visited on 07/30/2022).

[30] C. Richter, A. Bry, and N. Roy. “Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments”. In: *Springer International Publishing* 114 (Apr. 2016), pp. 649–666. ISSN: 1610-7438. URL: https://dspace.mit.edu/handle/1721.1/106840 (visited on 08/13/2022).

[31] rshaffer. *Drone Payloads Vs. Performance: A Balancing Act*. en-US. Apr. 2021. URL: https://consortiq.com/uas-resources/how-to-balance-your-drones-payload-performance (visited on 06/26/2023).

[32] G. Shi et al. “Meta-Adaptive Nonlinear Control: Theory and Algorithms”. In: *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*. Vol. 34. Sydney, Australia: arXiv, Oct. 2021. DOI: 10.48550/arXiv.2106.06098. URL: http://arxiv.org/abs/2106.06098 (visited on 05/23/2022).

[33] X. Shi et al. “Adaptive Nonlinear Control of Fixed-Wing VTOL with Airflow Vector Sensing”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2020, pp. 5321–5327. DOI: 10.1109/ICRA40945.2020.9197344. URL: http://arxiv.org/abs/2003.07558 (visited on 05/23/2022).

[34] J.-J. E. Slotine and W. Li. *Applied nonlinear control*. Englewood Cliffs, N.J: Prentice Hall, 1991. ISBN: 978-0-13-040890-7.

[35] J. Slump. “Home of Innovation”. In: Robotics Special by TU Delft (2017), p. 25. URL: https://issuu.com/tudelft-mediasolutions/docs/hoi-robotics (visited on 07/30/2022).

[36] S. Sun et al. “A Comparative Study of Nonlinear MPC and Differential-Flatness-Based Control for Quadrotor Agile Flight”. In: arXiv, Feb. 2022. DOI: 10.48550/arXiv.2109.01365. URL: http://arxiv.org/abs/2109.01365 (visited on 05/23/2022).

[37] J. Svacha, K. Mohta, and V. Kumar. “Improving quadrotor trajectory tracking by compensating for aerodynamic effects”. In: IEEE, June 2017, pp. 860–866. DOI: 10.1109/ICUAS.2017.7991501.

[38] E. Tal and S. Karaman. “Accurate Tracking of Aggressive Quadrotor Trajectories using Incremental Nonlinear Dynamic Inversion and Differential Flatness”. In: *IEEE Transactions on Control Systems Technology*. Vol. 29. 3. IEEE, June 2020, pp. 1203–1218. DOI: 10.48550/arXiv.1809.04048. URL: http://arxiv.org/abs/1809.04048 (visited on 05/23/2022).

[39] G. Torrente et al. “Data-Driven MPC for Quadrotors”. In: *IEEE Robotics and Automation Letters*. Vol. 6. 2. IEEE, Mar. 2021, pp. 3769–3776. DOI: 10.48550/arXiv.2102.05773. URL: http://arxiv.org/abs/2102.05773 (visited on 05/23/2022).

[40] P. Ventura Diaz and S. Yoon. “High-Fidelity Computational Aerodynamics of Multi-Rotor Unmanned Aerial Vehicles”. en. In: *2018 AIAA Aerospace Sciences Meeting*. Kissimmee, Florida: American Institute of Aeronautics and Astronautics, Jan. 2018. ISBN: 978-1-62410-524-1. DOI: 10.2514/6.2018-1266. URL: https://arc.aiaa.org/doi/10.2514/6.2018-1266 (visited on 05/17/2023).

[41] Y. Zhou, E.-J. Van Kampen, and Q. Chu. “Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control”. In: IMAV 2016, Oct. 2016.

[42] *Zipline - Instant Logistics*. Website. URL: https://www.flyzipline.com/ (visited on 07/26/2022).

Part III
Additional Work

A

Code Architecture

The onboard adaptive neural controller implementation in Paparazzi-UAV is available at https://github.com/MauroVA98/paparazzi-ANC while the offline DAIML training implementation in Pytorch is available at https://github.com/MauroVA98/training-ANC.

The code architecture of the adaptive neural controller implementation onboard the Parrot Bebop 1 is shown in Figure A.1. The code is implemented in a modular architecture comprised of four programs: $min\_snap.c$, $phi.c$, $adapt.c$ & $main.c$ corresponding to the trajectory generation, network inference, coefficient adaptation and the controller itself. The code implementation is easy-to-use yet versatile with all tunable parameters defined in the $init.h$ header file shown in Listing 1. Additionally, the network weights and biases and the trajectory polynomial coefficients are not hard-coded, instead they are loaded from $.txt$ files which allows for effortless changing of networks or trajectories on-the-fly.

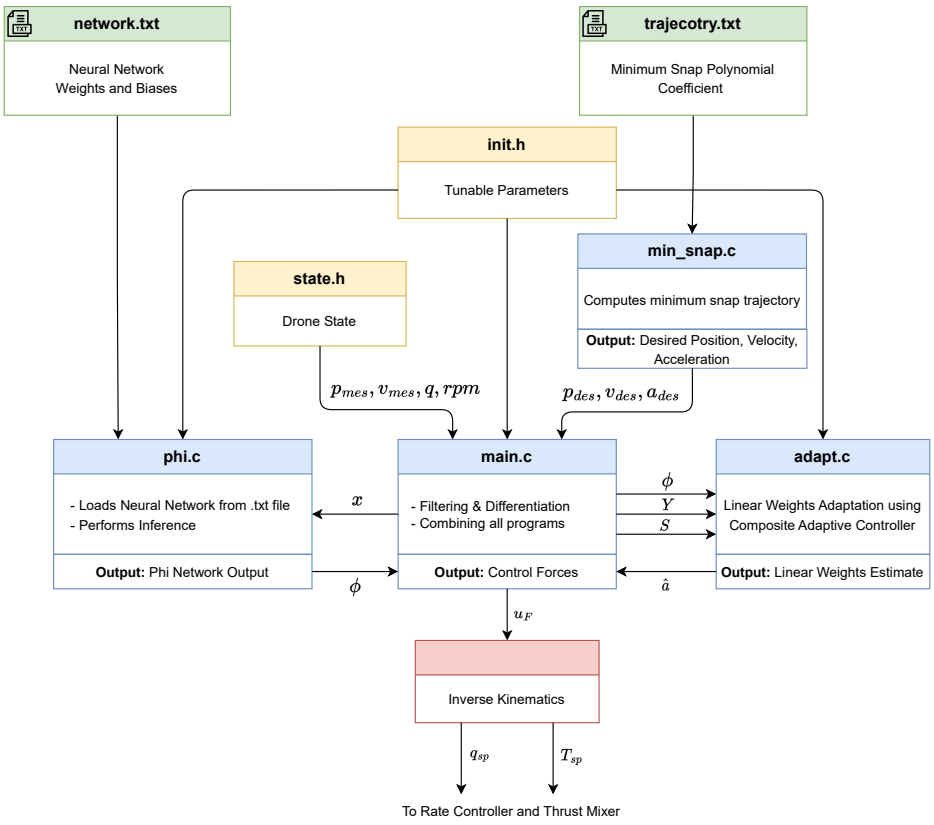


Figure A.1: Adaptive Neural Controller Code Architecture Diagram

For example, if the network output size PHI_OUT_DIM is increased not only will the network loading and inference work seamlessly with the new size but also the adaptation coefficient matrix size will change such that the output of the controller $\boldsymbol{\phi}(\boldsymbol{x})\, A$ remains a 3×1 vector of the unmodeled forces.

```
/// ADAPTIVE NEURAL CONTROLLER
#define P_TRIM -0.060                           // pitch trim [rad]
#define R_TRIM -0.002                           // roll trim  [rad]
static float K_P[3] = {0.2f, 0.2f, 1.0f};       // position feedback gain
static float K_V[3] = {1.0f, 1.0f, 2.0f};       // velocity feedback gain
static float f_cutoff[3] = {0.7f, 0.7f, 0.7f};  // butterworth cut-off frequency

// Thrust & Aerodynamic Model
static float k_t = 6.70e-08;
static float k_x = 1.08e-05;
static float k_y = 9.65e-06;
static float k_z = 2.79e-05;
static float k_h = 6.26e-02;


/// PHI NETWORK
static float RPM_ratio = 10000.0f;              // RPM normalization ratio

// Dimensions
#define PHI_IN_DIM 11                           // input dimensions
#define LAY1_DIM 50                             // layer 1 output dimensions
#define LAY2_DIM 60                             // layer 2 output dimensions
#define LAY3_DIM 50                             // layer 3 output dimensions
#define PHI_OUT_DIM 4                           // output dimensions

// Neural Network .txt file path (Drone & Sim)
#if AP == 1
#define NET_PATH "/data/ftp/internal_000/nn/03_30_19_31.txt"
#elif AP == 0
#define NET_PATH ".../03_30_19_31/600.txt"
#endif


/// ADAPTATION
#define ADAPT_IN_DIM PHI_OUT_DIM                // adaptation input dimension
#define ADAPT_MEAS_DIM 1                        // adaptation measurement dimensions
static float lambda = 0.01f;                    // damping coefficient @50Hz

// Adaptation Matrix Diagonal Values
static float q = 10.0f;                         // State Noise Covariance Matrix
static float r = 50.0f;                         // Measurement Noise Covariance Matrix
```

Listing 1: Sample Section of init.h Header File

B

OptiTrack Filtering

A zero-phase ”filtfilt” filter is used to remove noise in the OptiTrack velocity measurements. The quadrotor acceleration is then obtained through 4th-order central difference of the filtered velocity measurements, followed by computing the residual force acting on the quadrotor by solving Equation 5.2.

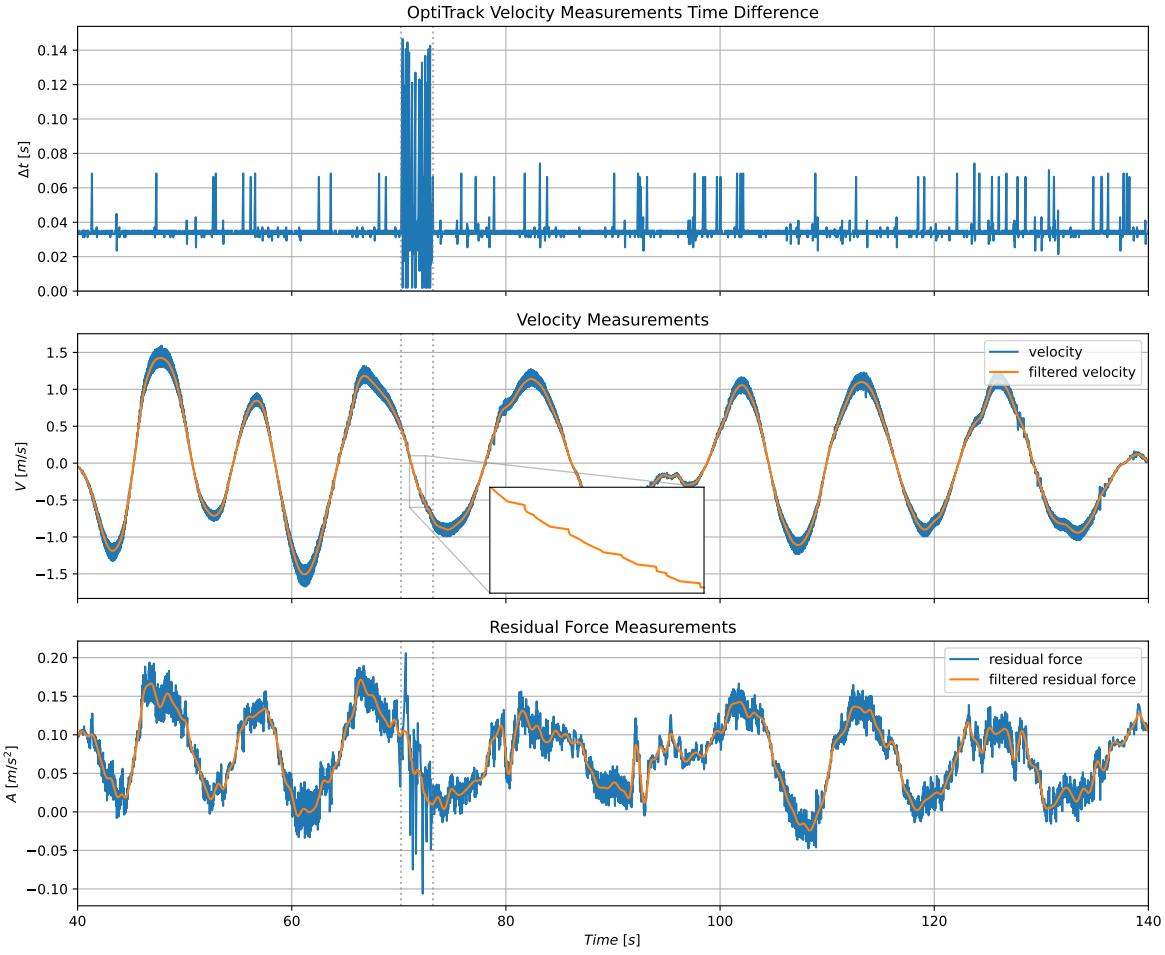


Figure B.1: ”FiltFilt” Zero-Phase Smoothing of OptiTrack Velocity Measurements and Residual Force

The filtered velocity signal removes most of the noise in the unfiltered velocity measurements without introducing lag. This always for the computation of closest possible ”ground-truth” of the residual

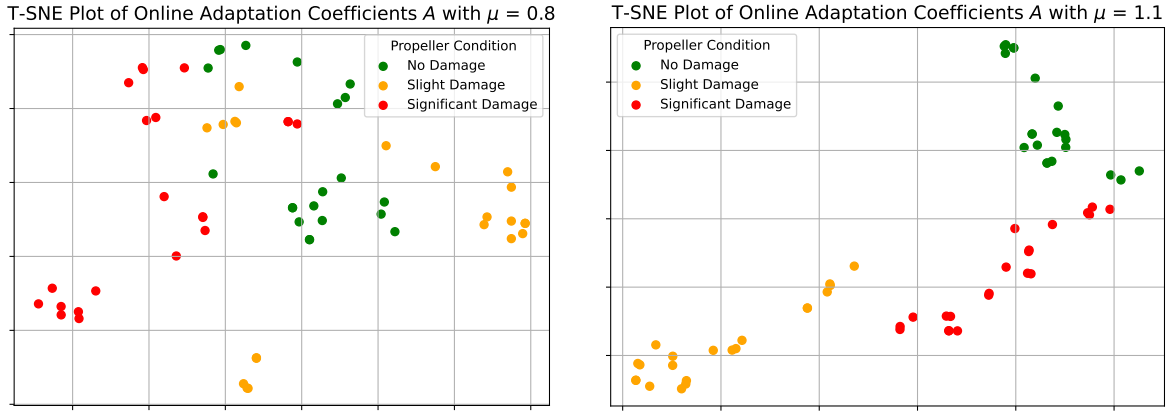
force. The quality of the residual force is mostly acceptable, however for seemingly random intervals the quality of the residual force significantly deteriorates. Upon further investigation it was found that this was caused by asynchronous OptiTrack measurements during this interval, with time differences between OptiTrack measurements considerably increasing as seen in Figure B.1. Given that the zero-phase filter assumes a constant dt the asynchronous measurements would result in bumps in the filtered velocity signal which are amplified when performing central difference to obtain acceleration.

A second zero-phase ”filtfilt” filter is applied on the acceleration signal to improve the quality of the residual force and thus also the predictions of the ϕ network without sacrificing valuable training data. The filtered residual force significantly reduces the differentiation noise during the asynchronous OptiTrack measurement intervals in addition to removing the unwanted high-frequency components.

C

Propeller Damage Estimation

An interesting feature of the adaptive neural control architecture is ability to estimate the in-flight propeller damage condition from the online adaptation coefficients $\hat{\boldsymbol{A}}$. The online adaptation coefficients will differ from those obtained during training due to the regularization and position tracking terms in the coefficient update in Equation 4.42. To determine wherever the online adaptation coefficients contain propeller condition information the high dimensional adaptation coefficients are displayed in a 2-D plane using t-SNE dimensionality reduction with speed factors $w = 0.7$ & $w = 1.1$ in Figure C.1a and Figure C.1b respectively.



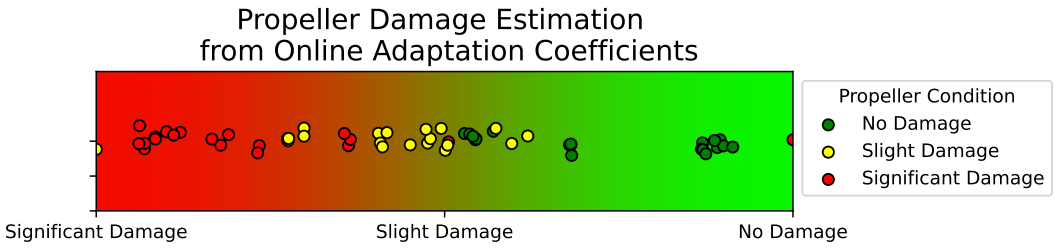
(a) with $w = 0.7$

(b) with $w = 1.1$

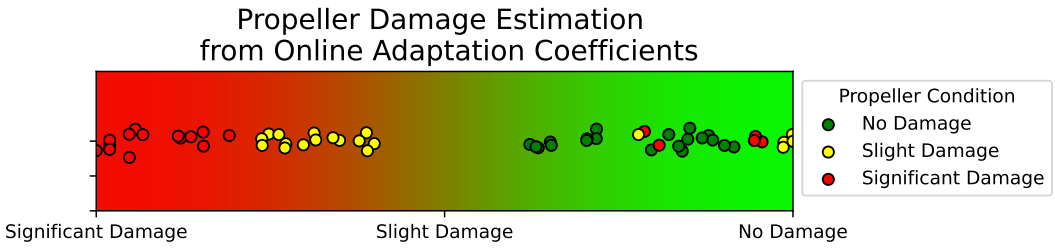
Figure C.1: t-SNE Plots of Online Adaptation Coefficients ($\hat{\boldsymbol{A}}$)

We notice in the t-SNE plots that the grouping of propeller damage conditions becomes more significant as the speed factor increases. Indicating that at higher speeds the propeller condition information is more dominant in the adaptation coefficients whereas at lower speeds the tracking error or source of noise are more dominant.

To better showcase the ability of estimating the propeller damage condition from the online adaptation coefficients t-SNE dimensionality reduction is performed to 1-D. These t-SNE plots are shown with speed factors $w = 0.7$ & $w = 1.1$ in Figure C.2a and Figure C.2b respectively.



(a) with $w = 0.8$



(b) with $w = 1.1$

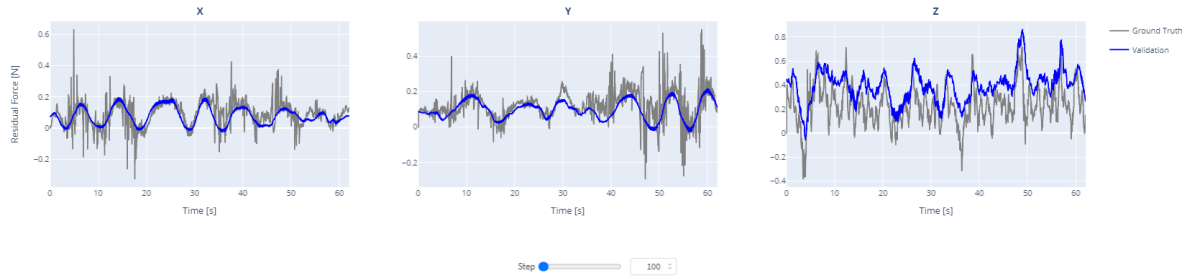
Figure C.2: Propeller Damage Condition Estimation from Online Adaptation Coefficients

It should be noted that t-SNE groups propeller conditions based on local similarities in the higher dimensional data with any random order, thus various t-SNE attempts where required to obtain the decreasing proper condition order shown (i.e. significant damage, slight damage & no damage). Additionally, t-SNE is an iterative algorithm for which the dimension reduction transformation is unknown but does show that it is possible. More advanced dimensionality reduction methods with known transformations would be required to deploy propeller condition estimation on-the-fly.

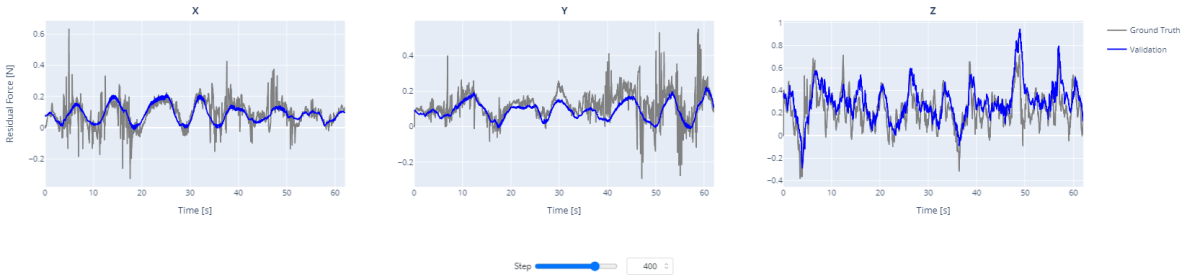
D

WandB Integration

Tracking the neural network training results using the Domain Adversarially Invariant Meta-Learning (DAIML) algorithm is usually a tedious and time-consuming task prone to errors. Thankfully, the DAIML training implementation has been integrated with WandB [1] a intelligence tracking and versioning tool for neural network experiments. In addition to the standard training and validation loss, WandB can track any plots, table and values throughout each training run. We use WandB to log the loss plots, t-SNE plots, force prediction plots, adaptation coefficients and test loss metrics at various epochs, for example the testing force prediction plots for $epoch = 100$ and $epoch = 400$ are shown in Figure D.1a & Figure D.1b. This allows us to easily see the training progress and its influence on the force predictions. Comparing results between runs in also straightforward as shown in Figure D.2. Additionally, WandB supports hyperparameter sweeps through a $.yaml$ file to determine the optimal hydrometers that minimize a tracked value.



(a) at $epoch = 100$



(b) at $epoch = 400$

Figure D.1: Residual Force Prediction Plots of Testing Data

[1] https://wandb.ai/site

Figure D.2: Comparing Training & Validation Loss of Multiple Runs in WandB