Methods for Robust Learning-Based Control

Thesis by
Michael O'Connell

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy in Space Engineering

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2023
Defended May 3rd, 2023

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Professor Soon-Jo Chung. Prof. Chung has been a great source of support, beginning at the University of Illinois, where he was my undergraduate academic and research advisor for most of my time there, through my time at Caltech working on my PhD. I would also like to thank a number of other professors who have provided guidance and support, including Professors Yisong Yue and Joel Burdick, who are on my defense committee, and Professor Anima Anandkumar. Thank you to Professor Sergio Pellegrino for serving on my defense committee and qualifying exam committee, and for always being ready to inquire about any student's research.

I would also like to thank my many collaborators, especially Drs. Xichen Shi, Guanya Shi, Matthew Anderson, and Kamyar Azizzadenesheli, and Patrick Spieler and Joshua Cho, for their insight, discussions, and willingness to hear out my sometimes barely-coherent ideas. I have also had a great number of external collaborators, who have provided valuable insights and feedback throughout my PhD, as well as supported the projects that I worked on. I would especially like to mention my collaborators at Jump Aero. This leads me to thanking the number of funding sources that have supported my research, including the Vought Fellowship, the Raytheon Company, Defense Advanced Research Projects Agency (DARPA), and Supernal.

Next I would like to thank many of the groups at Caltech that make the Caltech experience so unique. First, Michele Judd and the staff and affiliates of the Keck Institute for Space Studies, for providing invaluable opportunities and guidance for professional growth, and for providing a great community of students and researchers. Next, I would like to thank the Caltech Alpine Club for being an inspiring community of outdoor weekend warriors, and especially Dr. Jacqueline Dowling for being co-president of the club with me during and for being a great ski buddy, and all the other rad club members, with whom I have spent many hours wandering in the wilderness. And, I would like to thank the Caltech Wind Orchestra, especially Dr. Glenn Price, Professor Paul Asimow, and Professor Bill Bing, for providing a warm and welcoming community within the Caltech.

I would like to thank my friends, from all stages, who have provided support and encouragement, including my sousaphone+ friends, let's-road-trip-literally-across-the-country friends, fellow brewers and beer aficionados, the first year GALCIT cohort, and many others.

Finally, I would like to thank my family and friends for their support throughout my academic journey. My parents, Mike and Amy O'Connell, have always been supportive of my many interests and are ready to lend a keen ear for anything at all. My siblings Billy and Ellen O'Connell have always been great camping buddies, bike-shopping advisors, and friends, and our late dog, Riley, was an inspiring and accomplished bunny hunter.

The work presented in this thesis would not have been possible without the help and support of these people, and many others who I am surely forgetting, who have been supportive and encouraging through all the toughest parts of grad school.

ABSTRACT

This thesis addresses the general problem of improving control, safety, and reliability of multi-rotor drones in various challenging conditions by introducing novel deep-learning-based approaches. These approaches are designed to tackle specific issues that multi-rotor drones face during operation, such as near-ground trajectory control, high-speed wind disturbances, actuation delays, and motor failures. The thesis is organized into four main chapters, plus an introduction and conclusion. Each of the main chapters focuses on a unique approach to address a particular challenge of deep-learning-based control methods. Chapter 2 presents Neural-Lander, a deep-learning-based robust nonlinear controller that significantly improves quadrotor control performance during landing by accounting for complex aerodynamic effects. This chapter addresses key challenges to incorporating learned residual dynamics into a control architecture, laying the groundwork for the subsequent chapters. Chapters 3 and 4 introduce Neural-Fly, a learning-based approach that uses Domain Adversarially Invariant Meta-Learning (DAIML) and adaptive control to enable rapid online learning and precise flight control under a wide range of wind conditions. Chapter 5 proposes a lightweight augmentation method that enhances trajectory tracking performance for UAVs by effectively compensating for motor dynamics and digital transport delays. This method is extensible to a range of control methods, including learning-based approaches. Chapter 6 explores a novel sparse failure identification method for detecting and compensating for motor failures in over-actuated UAVs, contributing to the development of robust fault detection and compensation strategies for a safer and more reliable operation. This method builds on the Neural-Fly online learning framework and extends it to handle a wider range of conditions, including complete actuator failures. Together, these chapters address key challenges in safe and reliable learning-based control and demonstrate the potential of deep-learning-based control methods.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] M. O'Connell, G. Shi, X. Shi, *et al.*, "Neural-Fly enables rapid learning for agile flight in strong winds," *Science Robotics*, May 4, 2022. DOI: `10.1126/scirobotics.abm6597`. [Online]. Available: `https://www.science.org/doi/full/10.1126/scirobotics.abm6597` (visited on 05/13/2022), M.O. and G.S. contributed equally to this work, and they are both listed as first authors. M.O. led the design of the adaptive controller, substantially contributed to the design of the learning algorithm, led the hardware experiments, and substantially participated in writing the manuscript. This article is included in Chapters 3 and 4.

[2] G. Shi, K. Azizzadenesheli, M. O'Connell, S.-J. Chung, and Y. Yue, "Meta-adaptive nonlinear control: Theory and algorithms," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10 013–10 025, Dec. 6, 2021. [Online]. Available: `https://proceedings.neurips.cc/paper/2021/hash/52fc2aee802efbad698503d28ebd3a1f-Abstract.html` (visited on 03/27/2023),
M.O. helped propose and study the idea of adapting the last layer of the neural network, and conducted simulation validations for all the proposed methods.

[3] X. Shi, M. O'Connell, and S.-J. Chung, "Numerical predictive control for delay compensation," Sep. 30, 2020. arXiv: `2009.14450 [cs, eess]`. [Online]. Available: `http://arxiv.org/abs/2009.14450` (visited on 09/21/2021), M.O. implemented and conducted experiments for the proposed methods, and helped write the manuscript.

[4] G. Shi, X. Shi, M. O'Connell, *et al.*, "Neural lander: Stable drone landing control using learned dynamics," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 9784–9790. DOI: `10.1109/ICRA.2019.8794351`,
G.S., X.S., and M.O. contributed equally to this work. M.O. substantially contributed to the design of the method and led the hardware experiments. This article is included in Chapter 2.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

The use of uninhabited aerial vehicles (UAVs) has grown rapidly in recent years, with applications ranging from casual consumer camera drones to autonomous commercial delivery and transportation, and even high-tech stealth aircraft. Each of these applications presents a unique set of challenges, but all require precise control of the vehicle to achieve the desired performance. For instance, drone delivery necessitates reliable transportation of goods to residential and commercial areas in various weather conditions; drone-based first responder and search and rescue tasks demand safe navigation through remote wilderness settings, roadside accidents, and dense urban environments; and urban air mobility calls for flying cars to closely follow a planned path to avoid collisions with other vehicles in so-called urban canyons, which present numerous challenges to vehicle navigation and control.

The diverse set of UAV applications share a common requirement for absolute safety and reliability during operations, creating many challenges in their design, operation, and control. We will focus on several challenges specific to the control of UAVs. For example, the system must be able to operate in a wide range of conditions, such as high wind speeds, which is further complicated by complex urban environments that can create turbulent airflows. The system must also be able to operate in the presence of faults, such as a failure of an actuator, which not only must be quickly detected and compensated for to prevent total loss of control but also must be addressed to allow a safe landing. Finally, any control system must be integrated into a verifiable architecture; this often leads to a hierarchy of controllers, which enables each layer to be independently verified but results in computational overhead and delays in the control loop. In this thesis, we will focus on addressing these challenges through safe, robust, and agile learning-based control methods.

UAV control algorithms demonstrate impressive performance in controlled environments and on highly maneuverable systems, and UAV hardware is rapidly progressing. Agile flight control has been demonstrated on several systems, and impressive results have been achieved on a range of systems from quadcopters in laboratory environments [1], [2] to military aircraft in controlled testbeds [3]. Delivery drones and self-driving cars are being rolled out in limited environments, with companies such as Amazon running very early trials of their Prime Air delivery service [4] and Waymo deploying their self-driving cars after accumulating over 1 million miles of passenger-only autonomous driving [5]. Urban air mobility is still in the early stages of development, with many companies racing to develop small-scale electric aircraft to revolutionize urban transportation.

By examining this short list of current capabilities, we can identify several current limitations of state-of-the-practice and state-of-the-art advanced control systems. The most advanced and capable methods often rely on very fast and aggressive control methods. This results in a tightly coupled control hierarchy [6], which must be painstakingly designed and tuned for each vehicle configuration, operating condition, and application, or which relies on the underlying maneuverability of the vehicle to force the desired behavior from the system [7]. Either way, the design of control methods is further complicated by external factors such as difficult-to-navigate environments and weather conditions, and the likelihood of faults on large-scale systems. From these complexities, we see that the design space for control methods is both extremely large and necessarily large for the wide deployment of UAVs. However, the scale of the systems we are working with is also a great asset and opens the door to new methods of data-driven control. In this thesis, we will address problems presented by the complexity of the design space through a fusion of classic control methods and black-box learning methods, that is, methods which can leverage both physics-based models and data-driven methods to achieve robust and agile control of UAVs under a wide range of conditions.

In particular, this thesis will focus on a class of learning-based control methods which we call residual-learning-based control. This scheme concentrates on learning the difference between the desired and actual system response, or residual, and using that model to predict the system response to disturbances. This approach is similar to many recent adaptive control methods, which are formulated to quickly react to residual disturbances. However, by leveraging experience, in the form of recorded flight data, and the synthesizing power of machine learning, we can develop a model of the residual dynamics that can anticipate and proactively compensate for disturbances. This approach allows us to achieve a Goldilocks zone of control methods that are stable, robust, and agile while being able to learn and continuously improve the performance of the control system. The following four chapters discuss key considerations and challenges in the design of learning-based control systems for UAVs.

In Chapter 2, we tackle challenges of residual learning to counter unmodeled disturbances. Firstly, we show how to overcome non-affine control equations resulting from residual learning by using the previous control command as input to the neural network, solving for the new command using fixed-point iteration. Secondly, we demonstrate that a Lipschitz bounded learned model guarantees a control solution and enables the fixed-point iteration scheme. Finally, we provide stability analysis of the closed-loop system, showing that it is robust to errors in the learned model. We tested this residual learning framework on a quadrotor UAV during landing and near-object flights, where ground effect causes significant aerodynamic disturbances. This lays the groundwork for the following chapters, by establishing a framework for residual learning-based control and demonstrating its effectiveness in the presence of unmodeled disturbances.

In Chapter 3, we extend the offline learning of Chapter 2 to online learning of disturbances in varying exogenous conditions, with additional details discussed in Chapter 4. Our method is called Neural-Fly. Instead of learning the unmodeled disturbances as a neural network, we learn a representation of the disturbances as a set of basis functions that only depend on the endogenous factors such as the vehicle state. A novel formulation of meta-learning disentangles the effects of the exogenous conditions from the endogenous state so that only a set of mixing coefficients must be updated in new wind conditions. To update the model in real-time, we use a robust adaptive control law, which does not require persistent excitation of the representation and mitigates the effects of error in the learned model. We demonstrate this approach on a quadrotor UAV in a wind tunnel, where we show that Neural-Fly outperforms state-of-the-art control methods in mitigating wind disturbances while tracking an agile trajectory. This work demonstrates an effective approach to online learning of disturbances in varying exogenous conditions.

In Chapter 5, we show how to improve general controller performance in the presence of large delays. In particular, we address the problem of dead time delay resulting from digital control computation time and communication delays, and of physical delays in the system, such as the time it takes for a motor to spin up to the commanded speed. We find that a first-order compensation scheme not only approximates the ideal delay compensation but also significantly improves the robustness of the controller to both dead time and physical delays. We demonstrate this approach on a quadrotor UAV in a simulation environment, where we show that first-order delay compensation recovers the performance of the controller in the presence of large delays. Furthermore, this method is compatible with a variety of control methods, including residual learning-based control.

In Chapter 6, we examine two competing fault-tolerant control schemes and methods to incorporate residual learning for robust fault compensation. First, we look at residual learning-based fault-tolerant control, where the learned model is used to predict the fault and command a fault compensation. Although this method is effective, it relies on the availability of sufficient control bandwidth and lower-level control design to achieve the fault compensation. Second, we look at a fault-tolerant control scheme that builds on standard control schemes but computes a new control allocation that directly compensates for the fault. Both of these frameworks are tested on octocopter UAV platforms. Finally, we discuss some considerations for future work incorporating residual learning into these fault-tolerant control schemes.

This thesis explores the potential of residual-learning-based control methods to address the challenges of UAV control in complex environments and situations. By combining classic control methods with black-box learning approaches, we aim to achieve robust and agile control of UAVs under various conditions. Through extensive analysis and experimentation, we demonstrate the effectiveness of the proposed methods in overcoming unmodeled disturbances, adapting to varying exogenous conditions, compensating for large delays, and incorporating fault tolerance. By addressing these challenges, our work contributes to the ongoing development and large-scale deployment of UAVs for a wide range of applications, ultimately pushing the boundaries of what these versatile aerial platforms can achieve.

Chapter 2

NEURAL-LANDER: STABLE CONTROL USING BLACK-BOX RESIDUAL MODELS

Abstract

Precise near-ground trajectory control is difficult for multi-rotor drones, due to the complex aerodynamic effects caused by interactions between multi-rotor airflow and the environment. Conventional control methods often fail to properly account for these complex effects and fall short in accomplishing smooth landing. In this paper, we present a novel deep-learning-based robust nonlinear controller (*Neural-Lander*) that improves control performance of a quadrotor during landing. Our approach combines a nominal dynamics model with a Deep Neural Network (DNN) that learns high-order interactions. We apply spectral normalization (SN) to constrain the Lipschitz constant of the DNN. Leveraging this Lipschitz property, we design a nonlinear feedback linearization controller using the learned model and prove system stability with disturbance rejection. To the best of our knowledge, this is the first DNN-based nonlinear feedback controller with stability guarantees that can utilize arbitrarily large neural nets. Experimental results demonstrate that the proposed controller significantly outperforms a Baseline Nonlinear Tracking Controller in both landing and cross-table trajectory tracking cases. We also empirically show that the DNN generalizes well to unseen data outside the training domain.

2.1 Introduction

Unmanned Aerial Vehicles (UAVs) require high precision control of aircraft positions, especially during landing and take-off. This problem is challenging largely due to complex interactions of rotor and wing airflows with the ground. The aerospace community has long identified such ground effect that can cause an increased lift force and a reduced aerodynamic drag. These effects can be both helpful and disruptive in flight stability [1], and the complications are exacerbated with multiple rotors. Therefore, performing automatic landing of UAVs is risk-prone, and requires expensive high-precision sensors as well as carefully designed controllers.

Compensating for ground effect is a long-standing problem in the aerial robotics community. Prior work has largely focused on mathematical modeling (e.g., [2]) as part of system identification (ID). These models are later used to approximate aerodynamics forces during flights close to the ground and combined with controller design for feed-forward cancellation (e.g., [3]). However, existing theoretical ground effect models are derived based on steady-flow conditions, whereas most practical cases exhibit unsteady flow. Alternative approaches, such as integral or adaptive control methods, often suffer from slow response and delayed feedback. [4] employs Bayesian Optimization for open-air control but not for take-off/landing. Given these limitations, the precision of existing fully automated systems for UAVs are still insufficient for landing and take-off, thereby necessitating the guidance of a human UAV operator during those phases.

To capture complex aerodynamic interactions without overly-constrained by conventional modeling assumptions, we take a machine-learning (ML) approach to build a black-box ground effect model using Deep Neural Networks (DNNs). However, incorporating such models into a UAV controller faces three key challenges. First, it is challenging to collect sufficient real-world training data, as DNNs are notoriously data-hungry. Second, due to high-dimensionality, DNNs can be unstable and generate unpredictable output, which makes the system susceptible to instability in the feedback control loop. Third, DNNs are often difficult to analyze, which makes it difficult to design provably stable DNN-based controllers.

The aforementioned challenges pervade previous works using DNNs to capture high-order non-stationary dynamics. For example, [5], [6] use DNNs to improve system ID of helicopter aerodynamics, but not for controller design. Other approaches aim to generate reference inputs or trajectories from DNNs [7]–[10]. However, these approaches can lead to challenging optimization problems [7], or heavily rely on well-designed closed-loop controller and require a large number of labeled training data [8]–[10]. A more classical approach of using DNNs is direct inverse control [11]–[13] but the non-parametric nature of a DNN controller also makes it challenging to guarantee stability and robustness to noise. [14] proposes a provably stable model-based Reinforcement Learning method based on Lyapunov analysis, but it requires a potentially expensive discretization step and relies on the native Lipschitz constant of the DNN.

Contributions. In this paper, we propose a learning-based controller, *Neural-Lander*, to improve the precision of quadrotor landing with guaranteed stability. Our approach directly learns the ground effect on coupled unsteady aerodynamics and vehicular dynamics. We use deep-learning for system ID of residual dynamics and then integrate it with nonlinear feedback linearization control.

We train DNNs with layer-wise spectrally normalized weight matrices. We prove that the resulting controller is globally exponentially stable under bounded learning errors. This is achieved by exploiting the Lipschitz bound of spectrally normalized DNNs. It has earlier been shown that spectral normalization of DNNs leads to good generalization, i.e., stability in a learning-theoretic sense [15]. It is intriguing that spectral normalization simultaneously guarantees stability both in a learning-theoretic and a control-theoretic sense.

We evaluate *Neural-Lander* on trajectory tracking of quadrotor during take-off, landing and cross-table maneuvers. *Neural-Lander* is able to land a quadrotor much more accurately than a Baseline Nonlinear Tracking Controller with a pre-identified system. In particular, we show that compared to the baseline, *Neural-Lander* can decrease error in z axis from 0.13 m to 0, mitigate x and y drifts by as much as 90%, in the landing case. Meanwhile, *Neural-Lander* can decrease z error from 0.153 m to 0.027 m, in the cross-table trajectory tracking task.[1] We also demonstrate that the learned model can handle temporal dependency, and is an improvement over the steady-state theoretical models.

2.2 Problem Statement: Quadrotor Landing

Given quadrotor states as global position $\mathbf{p} \in \mathbb{R}^3$, velocity $\mathbf{v} \in \mathbb{R}^3$, attitude rotation matrix $R \in \mathrm{SO}(3)$, and body angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$, we consider the following dynamics:

$$\dot{\mathbf{p}} = \mathbf{v}, \qquad m\dot{\mathbf{v}} = m\mathbf{g} + R\mathbf{f}_u + \mathbf{f}_a, \tag{2.1a}$$

$$\dot{R} = RS(\boldsymbol{\omega}), \qquad J\dot{\boldsymbol{\omega}} = J\boldsymbol{\omega} \times \boldsymbol{\omega} + \boldsymbol{\tau}_u + \boldsymbol{\tau}_a, \tag{2.1b}$$

where m and J are the mass and inertia matrices of the system, respectively, $S(\cdot)$ is skew-symmetric mapping. $\mathbf{g} = [0, 0, -g]^\top$ is the gravity vector, $\mathbf{f}_u = [0, 0, T]^\top$ and $\boldsymbol{\tau}_u = [\tau_x, \tau_y, \tau_z]^\top$ are the total thrust and body torques from four rotors predicted by a nominal model. We use $\boldsymbol{\eta} = [T, \tau_x, \tau_y, \tau_z]^\top$ to denote the output wrench. Typical quadrotor control input uses squared motor speeds $\mathbf{u} = [n_1^2, n_2^2, n_3^2, n_4^2]^\top$, and is linearly related to the output wrench $\boldsymbol{\eta} = B_0\mathbf{u}$, with

$$B_0 = \begin{bmatrix} c_T & c_T & c_T & c_T \\ 0 & c_T l_{\text{arm}} & 0 & -c_T l_{\text{arm}} \\ -c_T l_{\text{arm}} & 0 & c_T l_{\text{arm}} & 0 \\ -c_Q & c_Q & -c_Q & c_Q \end{bmatrix}, \tag{2.2}$$

where c_T and c_Q are rotor force and torque coefficients, and l_{arm} denotes the length of rotor arm. The key difficulty of precise landing is the influence of unknown disturbance forces $\mathbf{f}_a = [f_{a,x}, f_{a,y}, f_{a,z}]^\top$ and torques $\boldsymbol{\tau}_a = [\tau_{a,x}, \tau_{a,y}, \tau_{a,z}]^\top$, which

[1]Demo videos: `https://youtu.be/FLLsG0S78ik`

originate from complex aerodynamic interactions between the quadrotor and the environment.

Problem Statement: We aim to improve controller accuracy by learning the unknown disturbance forces $\mathbf{f}_a$ and torques $\boldsymbol{\tau}_a$ in (2.1). As we mainly focus on landing and take-off tasks, the attitude dynamics is limited and the aerodynamic disturbance torque $\boldsymbol{\tau}_a$ is bounded. Thus position dynamics (2.1a) and $\mathbf{f}_a$ will our primary concern. We first approximate $\mathbf{f}_a$ using a DNN with spectral normalization to guarantee its Lipschitz constant, and then incorporate the DNN in our exponentially-stabilizing controller. Training is done off-line, and the learned dynamics is applied in the on-board controller in real-time to achieve smooth landing and take-off.

2.3 Dynamics Learning using DNN

We learn the unknown disturbance force $\mathbf{f}_a$ using a DNN with Rectified Linear Units (ReLU) activation. In general, DNNs equipped with ReLU converge faster during training, demonstrate more robust behavior with respect to changes in hyperparameters, and have fewer vanishing gradient problems compared to other activation functions such as *sigmoid* [16].

ReLU Deep Neural Networks

A ReLU deep neural network represents the functional mapping from the input $\mathbf{x}$ to the output $f(\mathbf{x}, \boldsymbol{\theta})$, parameterized by the DNN weights $\boldsymbol{\theta} = W^1, \cdots, W^{L+1}$:

$$f(\mathbf{x}, \boldsymbol{\theta}) = W^{L+1}\phi(W^L(\phi(W^{L-1}(\cdots\phi(W^1\mathbf{x})\cdots)))), \tag{2.3}$$

where the activation function $\phi(\cdot) = \max(\cdot, 0)$ is called the element-wise ReLU function. ReLU is less computationally expensive than *tanh* and *sigmoid* because it involves simpler mathematical operations. However, deep neural networks are usually trained by first-order gradient-based optimization, which is highly sensitive on the curvature of the training objective and can be unstable [17]. To alleviate this issue, we apply the spectral normalization technique [15].

Spectral Normalization

Spectral normalization stabilizes DNN training by constraining the Lipschitz constant of the objective function. Spectrally normalized DNNs have also been shown to generalize well [18], which is an indication of stability in machine learning. Mathematically, the Lipschitz constant of a function $\|f\|_{\text{Lip}}$ is defined as the smallest value such that

$$\forall\, \mathbf{x}, \mathbf{x}' : \; \|f(\mathbf{x}) - f(\mathbf{x}')\|_2 / \|\mathbf{x} - \mathbf{x}'\|_2 \leq \|f\|_{\text{Lip}}.$$

It is known that the Lipschitz constant of a general differentiable function f is the maximum spectral norm (maximum singular value) of its gradient over its domain $\|f\|_{\text{Lip}} = \sup_{\mathbf{x}} \sigma(\nabla f(\mathbf{x}))$.

The ReLU DNN in (2.3) is a composition of functions. Thus we can bound the Lipschitz constant of the network by constraining the spectral norm of each layer $g^l(\mathbf{x}) = \phi(W^l\mathbf{x})$. Therefore, for a linear map $g(\mathbf{x}) = W\mathbf{x}$, the spectral norm of each layer is given by $\|g\|_{\text{Lip}} = \sup_{\mathbf{x}} \sigma(\nabla g(\mathbf{x})) = \sup_{\mathbf{x}} \sigma(W) = \sigma(W)$. Using the fact that the Lipschitz norm of ReLU activation function $\phi(\cdot)$ is equal to 1, with the inequality $\|g_1 \circ g_2\|_{\text{Lip}} \leq \|g_1\|_{\text{Lip}} \cdot \|g_2\|_{\text{Lip}}$, we can find the following bound on $\|f\|_{\text{Lip}}$:

$$\|f\|_{\text{Lip}} \leq \|g^{L+1}\|_{\text{Lip}} \cdot \|\phi\|_{\text{Lip}} \cdots \|g^1\|_{\text{Lip}} = \prod_{l=1}^{L+1} \sigma(W^l). \tag{2.4}$$

In practice, we can apply spectral normalization to the weight matrices in each layer during training as follows:

$$\bar{W} = W/\sigma(W) \cdot \gamma^{\frac{1}{L+1}}, \tag{2.5}$$

where γ is the intended Lipschitz constant for the DNN. The following lemma bounds the Lipschitz constant of a ReLU DNN with spectral normalization.

Lemma 2.3.1. *For a multi-layer ReLU network $f(\mathbf{x}, \boldsymbol{\theta})$, defined in (2.3) without an activation function on the output layer. Using spectral normalization, the Lipschitz constant of the entire network satisfies:*

$$\|f(\mathbf{x}, \bar{\boldsymbol{\theta}})\|_{Lip} \leq \gamma,$$

with spectrally-normalized parameters $\bar{\boldsymbol{\theta}} = \bar{W}^1, \cdots, \bar{W}^{L+1}$.

Proof. As in (2.4), the Lipschitz constant can be written as a composition of spectral norms over all layers. The proof follows from the spectral norms constrained as in (2.5). □

Constrained Training

We apply gradient-based optimization to train the ReLU DNN with a bounded Lipschitz constant. Estimating $\mathbf{f}_a$ in (2.1) boils down to optimizing the parameters $\boldsymbol{\theta}$ in the ReLU network in (2.3), given the observed value of $\mathbf{x}$ and the target output. In particular, we want to control the Lipschitz constant of the ReLU network.

The optimization objective is as follows, where we minimize the prediction error with constrained Lipschitz constant:

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{minimize}} \quad & \sum_{t=1}^{T} \frac{1}{T} \|\mathbf{y}_t - f(\mathbf{x}_t, \boldsymbol{\theta})\|_2 \\ \text{subject to} \quad & \|f\|_{\text{Lip}} \leq \gamma. \end{aligned} \tag{2.6}$$

In our case, $\mathbf{y}_t$ is the observed disturbance force and $\mathbf{x}_t$ is the observed state and control input. According to the upper bound in (2.4), we can substitute the constraint by minimizing the spectral norm of the weights in each layer. We use stochastic gradient descent (SGD) to optimize (2.6) and apply spectral normalization to regulate the weights. From Lemma 2.3.1, the trained ReLU DNN has a Lipschitz constant.

2.4 Neural-Lander Controller Design

Our *Neural-Lander* controller for 3-D trajectory tracking is constructed as a nonlinear feedback linearization controller whose stability guarantees are obtained using the spectral normalizaion of the DNN-based ground-effect model. We then exploit the Lipschitz property of the DNN to solve for the resulting control input using fixed-point iteration.

Reference Trajectory Tracking

The position tracking error is defined as $\tilde{\mathbf{p}} = \mathbf{p} - \mathbf{p}_d$. Our controller uses a composite variable $\mathbf{s} = 0$ as a manifold on which $\tilde{\mathbf{p}}(t) \to 0$ exponentially:

$$\mathbf{s} = \dot{\tilde{\mathbf{p}}} + \Lambda \tilde{\mathbf{p}} = \dot{\mathbf{p}} - \mathbf{v}_r \tag{2.7}$$

with Λ as a positive definite or diagonal matrix. Now the trajectory tracking problem is transformed to tracking a reference velocity $\mathbf{v}_r = \dot{\mathbf{p}}_d - \Lambda \tilde{\mathbf{p}}$.

Using the methods described in Sec. 2.3, we define $\hat{\mathbf{f}}_a(\boldsymbol{\zeta}, \mathbf{u})$ as the DNN approximation to the disturbance aerodynamic forces, with $\boldsymbol{\zeta}$ being the partial states used as input features to the network. We design the total desired rotor force $\mathbf{f}_d$ as

$$\mathbf{f}_d = (R\mathbf{f}_u)_d = \bar{\mathbf{f}}_d - \hat{\mathbf{f}}_a, \text{with } \bar{\mathbf{f}}_d = m\dot{\mathbf{v}}_r - K_v \mathbf{s} - m\mathbf{g}. \tag{2.8}$$

Substituting (2.8) into (2.1), the closed-loop dynamics would simply become $m\dot{\mathbf{s}} + K_v \mathbf{s} = \boldsymbol{\epsilon}$, with approximation error $\boldsymbol{\epsilon} = \mathbf{f}_a - \hat{\mathbf{f}}_a$. Hence, $\tilde{\mathbf{p}}(t) \to \mathbf{0}$ globally and exponentially with bounded error, as long as $\|\boldsymbol{\epsilon}\|$ is bounded [19]–[21].

Consequently, desired total thrust T_d and desired force direction $\hat{k}_d$ can be computed as

$$T_d = \mathbf{f}_d \cdot \hat{k}, \text{ and } \hat{k}_d = \mathbf{f}_d / \|\mathbf{f}_d\|, \tag{2.9}$$

with $\hat{k}$ being the unit vector of rotor thrust direction (typically z-axis in quadrotors). Using $\hat{k}_d$ and fixing a desired yaw angle, desired attitude R_d can be deduced [22]. We assume that a nonlinear attitude controller uses the desired torque $\boldsymbol{\tau}_d$ from rotors to track $R_d(t)$. One such example is in [21]:

$$\boldsymbol{\tau}_d = J\dot{\boldsymbol{\omega}}_r - J\boldsymbol{\omega} \times \boldsymbol{\omega}_r - K_\omega(\boldsymbol{\omega} - \boldsymbol{\omega}_r), \tag{2.10}$$

where the reference angular rate $\boldsymbol{\omega}_r$ is designed similar to (2.7), so that when $\boldsymbol{\omega} \to \boldsymbol{\omega}_r$, exponential trajectory tracking of a desired attitude $R_d(t)$ is guaranteed within some bounded error in the presence of bounded disturbance torques.

Learning-based Discrete-time Nonlinear Controller

From (2.2), (2.9) and (2.10), we can relate the desired wrench $\boldsymbol{\eta}_d = [T_d, \boldsymbol{\tau}_d^\top]^\top$ with the control signal $\mathbf{u}$ through

$$B_0\mathbf{u} = \boldsymbol{\eta}_d = \begin{bmatrix} \left(\bar{\mathbf{f}}_d - \hat{\mathbf{f}}_a(\boldsymbol{\zeta}, \mathbf{u})\right) \cdot \hat{k} \\ \boldsymbol{\tau}_d \end{bmatrix}. \tag{2.11}$$

Because of the dependency of $\hat{\mathbf{f}}_a$ on $\mathbf{u}$, the control synthesis problem here is non-affine. Therefore, we propose the following fixed-point iteration method for solving (2.11):

$$\mathbf{u}_k = B_0^{-1}\boldsymbol{\eta}_d\left(\mathbf{u}_{k-1}\right), \tag{2.12}$$

where $\mathbf{u}_k$ and $\mathbf{u}_{k-1}$ are the control input for current and previous time-step in the discrete-time controller. Next, we prove the stability of the system and convergence of the control inputs in (2.12).

2.5 Nonlinear Stability Analysis

The closed-loop tracking error analysis provides a direct correlation on how to tune the neural network and controller parameter to improve control performance and robustness.

Control Allocation as Contraction Mapping

We first show that the control input $\mathbf{u}_k$ converges to the solution of (2.11) when all states are fixed.

Lemma 2.5.1. *Define mapping $\mathbf{u}_k = \mathcal{F}(\mathbf{u}_{k-1})$ based on (2.12) and fix all current states:*

$$\mathcal{F}(\mathbf{u}) = B_0^{-1}\begin{bmatrix} \left(\bar{\mathbf{f}}_d - \hat{\mathbf{f}}_a(\boldsymbol{\zeta}, \mathbf{u})\right) \cdot \hat{k} \\ \boldsymbol{\tau}_d \end{bmatrix}. \tag{2.13}$$

If $\hat{\mathbf{f}}_a(\boldsymbol{\zeta}, \mathbf{u})$ is L_a-Lipschitz continuous, and $\sigma(B_0^{-1}) \cdot L_a < 1$; then $\mathcal{F}(\cdot)$ is a contraction mapping, and $\mathbf{u}_k$ converges to unique solution of $\mathbf{u}^* = \mathcal{F}(\mathbf{u}^*)$.

Proof. $\forall\, \mathbf{u}_1, \mathbf{u}_2 \in \mathcal{U}$ with $\mathcal{U}$ being a compact set of feasible control inputs; and given fixed states as $\bar{\mathbf{f}}_d$, $\boldsymbol{\tau}_d$ and $\hat{k}$, then:

$$\begin{aligned} \|\mathcal{F}(\mathbf{u}_1) - \mathcal{F}(\mathbf{u}_2)\|_2 &= \left\| B_0^{-1} \left(\hat{\mathbf{f}}_a(\boldsymbol{\zeta}, \boldsymbol{u}_1) - \hat{\mathbf{f}}_a(\boldsymbol{\zeta}, \boldsymbol{u}_2) \right) \right\|_2 \\ &\le \sigma(B_0^{-1}) \cdot L_a \, \|\mathbf{u}_1 - \mathbf{u}_2\|_2 \,. \end{aligned}$$

Thus, $\exists\, \alpha < 1$, s.t $\|\mathcal{F}(\mathbf{u}_1) - \mathcal{F}(\mathbf{u}_2)\|_2 < \alpha \, \|\mathbf{u}_1 - \mathbf{u}_2\|_2$. Hence, $\mathcal{F}(\cdot)$ is a contraction mapping. □

Stability of Learning-based Nonlinear Controller

Before continuing to prove the stability of the full system, we make the following assumptions.

Assumption 1. *The desired states along the position trajectory $\mathbf{p}_d(t)$, $\dot{\mathbf{p}}_d(t)$, and $\ddot{\mathbf{p}}_d(t)$ are bounded.*

Assumption 2. *One-step difference of control signal satisfies $\|\mathbf{u}_k - \mathbf{u}_{k-1}\| \le \rho \, \|\mathbf{s}\|$ with a small positive ρ.*

Here we provide the intuition behind this assumption. From (2.13), we can derive the following approximate relation with $\Delta(\cdot)_k = \|(\cdot)_k - (\cdot)_{k-1}\|$:

$$\begin{aligned} \Delta u_k \le \sigma(B_0^{-1}) \big(& L_a \Delta u_{k-1} + L_a \Delta \zeta_k \\ & + m \Delta \dot{v}_{r,k} + \lambda_{\max}(K_v) \Delta s_k + \Delta \tau_{d,k} \big). \end{aligned}$$

Because update rate of attitude controller ($> 100\,\text{Hz}$) and motor speed control ($> 5\,\text{kHz}$) are much higher than that of the position controller ($\approx 10\,\text{Hz}$), in practice, we can safely neglect Δs_k, $\Delta \dot{v}_{r,k}$, and $\Delta \zeta_k$ in one update (Theorem 11.1 [23]). Furthermore, $\Delta \tau_{d,k}$ can be limited internally by the attitude controller. It leads to:

$$\Delta u_k \le \sigma(B_0^{-1}) \big(L_a \Delta u_{k-1} + c \big),$$

with c being a small constant and $\sigma(B_0^{-1}) \cdot L_a < 1$ from Lemma. 2.5.1, we can deduce that Δu rapidly converges to a small ultimate bound between each position controller update.

Assumption 3. *The learning error of $\hat{\mathbf{f}}_a(\boldsymbol{\zeta}, \mathbf{u})$ over the compact sets $\boldsymbol{\zeta} \in \mathcal{Z}$, $\mathbf{u} \in \mathcal{U}$ is upper bounded by $\epsilon_m = \sup_{\boldsymbol{\zeta} \in \mathcal{Z}, \mathbf{u} \in \mathcal{U}} \|\boldsymbol{\epsilon}(\boldsymbol{\zeta}, \mathbf{u})\|$, where $\boldsymbol{\epsilon}(\boldsymbol{\zeta}, \mathbf{u}) = \mathbf{f}_a(\boldsymbol{\zeta}, \mathbf{u}) - \hat{\mathbf{f}}_a(\boldsymbol{\zeta}, \mathbf{u})$.*

DNNs have been shown to generalize well to the set of unseen events that are from almost the same distribution as training set [24], [25]. This empirical observation is also theoretically studied in order to shed more light toward an understanding of the complexity of these models [18], [26]–[28]. Based on the above assumptions, we can now present our overall stability and robustness result.

Theorem 2.5.2. *Under Assumptions 1-3, for a time-varying* $\mathbf{p}_d(t)$*, the controller defined in (2.8) and (2.12) with* $\lambda_{\min}(K_v) > L_a\rho$ *achieves exponential convergence of composite variable* $\mathbf{s}$ *to error ball* $\lim_{t\to\infty}\|\mathbf{s}(t)\| = \epsilon_m/(\lambda_{\min}(K_v) - L_a\rho)$ *with rate* $((\lambda_{\min}(K_v) - L_a\rho)/m$*. And* $\tilde{\mathbf{p}}$ *exponentially converges to error ball*

$$\lim_{t\to\infty}\|\tilde{\mathbf{p}}(t)\| = \frac{\epsilon_m}{\lambda_{\min}(\Lambda)(\lambda_{\min}(K_v) - L_a\rho)} \tag{2.14}$$

with rate $\lambda_{\min}(\Lambda)$*.*

Proof. We begin the proof by selecting a Lyapunov function as $\mathcal{V}(\mathbf{s}) = \frac{1}{2}m\|\mathbf{s}\|^2$, then by applying the controller (2.8), we get the time-derivative of $\mathcal{V}$:

$$\begin{aligned}\dot{\mathcal{V}} &= \mathbf{s}^\top\big(-K_v\mathbf{s} + \hat{\mathbf{f}}_a(\zeta_k,\mathbf{u}_k) - \hat{\mathbf{f}}_a(\zeta_k,\mathbf{u}_{k-1}) + \boldsymbol{\epsilon}(\zeta_k,\mathbf{u}_k)\big)\\ &\le -\mathbf{s}^\top K_v\mathbf{s} + \|\mathbf{s}\|(\|\hat{\mathbf{f}}_a(\zeta_k,\mathbf{u}_k) - \hat{\mathbf{f}}_a(\zeta_k,\mathbf{u}_{k-1})\| + \epsilon_m).\end{aligned}$$

Let $\lambda = \lambda_{\min}(K_v)$ denote the minimum eigenvalue of the positive-definite matrix K_v. By applying the Lipschitz property of $\hat{\mathbf{f}}_a$ Lemma 2.3.1 and Assumption 2, we obtain

$$\dot{\mathcal{V}} \le -\frac{2\,(\lambda - L_a\rho)}{m}\mathcal{V} + \sqrt{\frac{2\mathcal{V}}{m}}\epsilon_m.$$

Using the Comparison Lemma [23], we define $\mathcal{W}(t) = \sqrt{\mathcal{V}(t)} = \sqrt{m/2}\|\mathbf{s}\|$ and $\dot{\mathcal{W}} = \dot{\mathcal{V}}/\left(2\sqrt{\mathcal{V}}\right)$ to obtain

$$\|\mathbf{s}(t)\| \le \|\mathbf{s}(t_0)\|\exp\left(-\frac{\lambda - L_a\rho}{m}(t - t_0)\right) + \frac{\epsilon_m}{\lambda - L_a\rho}.$$

It can be shown that this leads to finite-gain $\mathcal{L}_p$ stability and input-to-state stability (ISS) [29]. Furthermore, the hierarchical combination between $\mathbf{s}$ and $\tilde{\mathbf{p}}$ in (2.7) results in $\lim_{t\to\infty}\|\tilde{\mathbf{p}}(t)\| = \lim_{t\to\infty}\|\mathbf{s}(t)\|/\lambda_{\min}(\Lambda)$, yielding (2.14). □

2.6 Experiments

In our experiments, we evaluate both the generalization performance of our DNN as well as the overall control performance of *Neural-Lander*. The experimental setup is

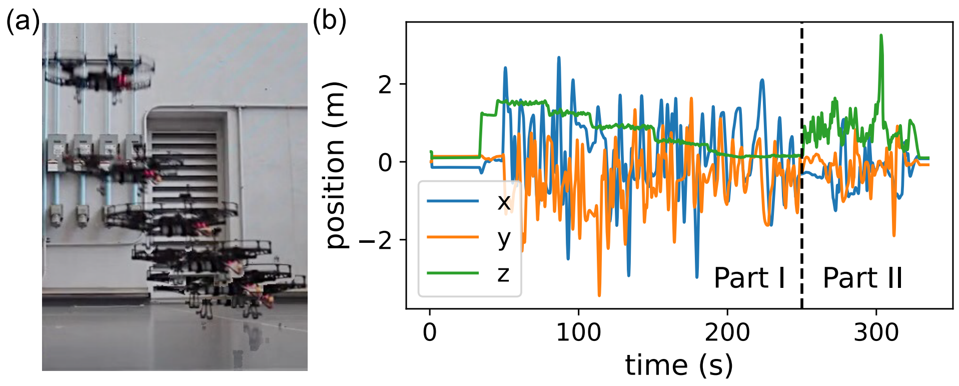


Figure 2.1: **Composite image of landing sequence and plot of training data.** (a) Composite image of an Intel Aero drone landing sequence; (b) Training data trajectory. Part I (0 to 250 s) contains maneuvers at different heights (0.05 m to 1.50 m). Part II (250 s to 350 s) includes random x, y, and z motions for maximum state-space coverage.

composed of a motion capture system with 17 cameras, a WiFi router for communication, and an Intel Aero drone, weighing 1.47 kg with an onboard Linux computer (2.56 GHz Intel Atom x7 processor, 4 GB DDR3 RAM). We retrofitted the drone with eight reflective infrared markers for accurate position, attitude and velocity estimation at 100Hz. The Intel Aero drone and the test space are shown in Fig. 2.1(a).

Bench Test

To identify a good nominal model, we first measured the mass, m, diameter of the rotor, D, the air density, ρ, gravity, g. Then we performed bench test to determine the thrust constant, c_T, as well as the non-dimensional thrust coefficient $C_T = \frac{c_T}{\rho D^4}$. Note that C_T is a function of propeller speed n, and here we picked a nominal value at $n = 2000\,\mathrm{RPM}$.

Real-World Flying Data and Preprocessing

To estimate the disturbance force $\mathbf{f}_a$, an expert pilot manually flew the drone at different heights, and we collected training data consisting of sequences of state estimates and control inputs $\{(\mathbf{p}, \mathbf{v}, R, \mathbf{u}), \mathbf{y}\}$ where $\mathbf{y}$ is the observed value of $\mathbf{f}_a$. We utilized the relation $\mathbf{f}_a = m\dot{\mathbf{v}} - m\mathbf{g} - R\mathbf{f}_u$ from (2.1) to calculate $\mathbf{f}_a$, where $\mathbf{f}_u$ is calculated based on the nominal c_T from the bench test in Sec. 2.6. Our training set is a single continuous trajectory with varying heights and velocities. The trajectory has two parts shown in Fig. 2.1(b). We aim to learn the ground effect through Part I of the training set, and other aerodynamics forces such as air drag through Part II.

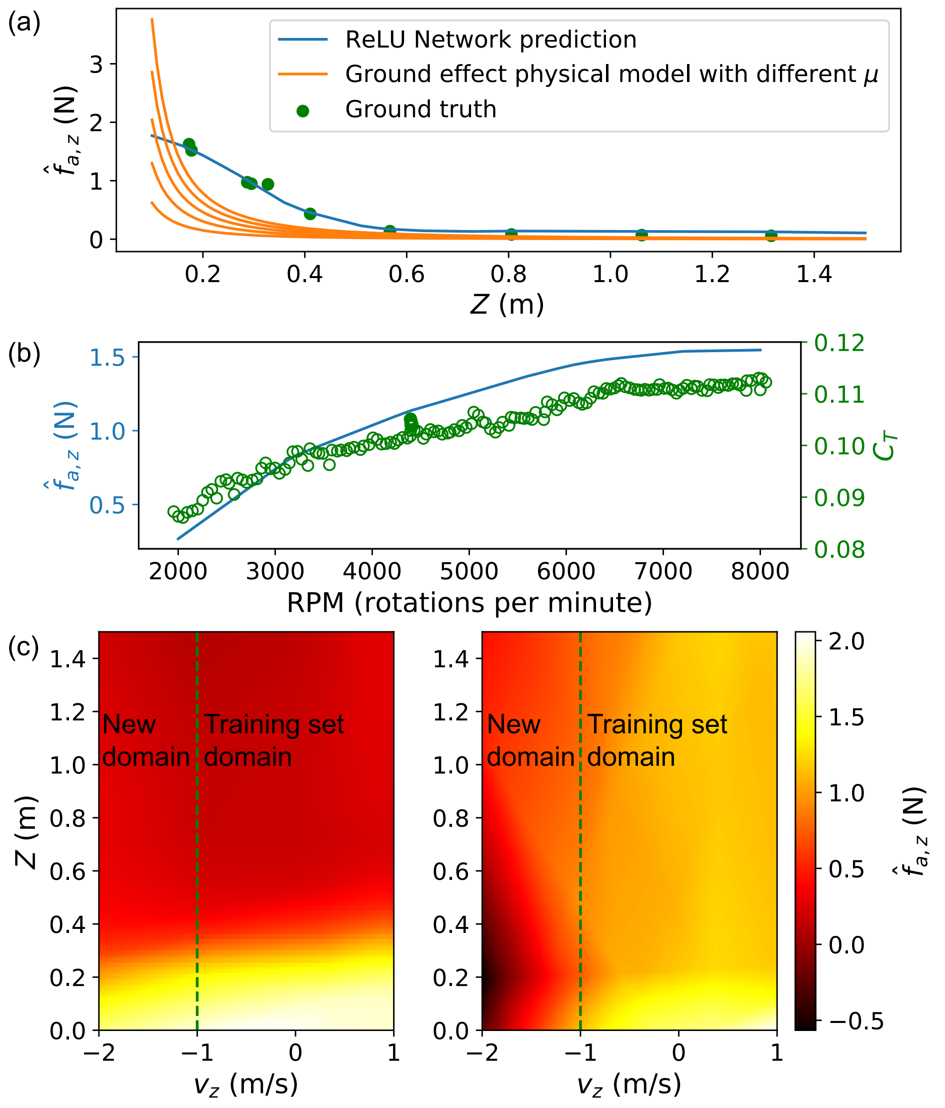


Figure 2.2: **Visualization of ground effect prediction performance.** (a) Learned $\hat{f}_{a,z}$ compared to the ground effect model with respect to height z, with $v_z = v_x = v_y = 0\,\text{m/s}$, $R = I$, $\mathbf{u} = 6400\,\text{RPM}$. Ground truth points are from hovering data at different heights. (b) Learned $\hat{f}_{a,z}$ with respect to rotation speed n ($z = 0.2\,\text{m}$, $v_z = 0\,\text{m/s}$), compared to C_T measured in the bench test. (c) Heatmaps of learned $\hat{f}_{a,z}$ versus z and v_z. (Left) ReLU network with spectral normalization. (Right) ReLU network without spectral normalization.

DNN Prediction Performance

We train a deep ReLU network $\hat{\mathbf{f}}_a(\zeta, \mathbf{u}) = \hat{\mathbf{f}}_a(z, \mathbf{v}, R, \mathbf{u})$, with z, $\mathbf{v}$, R, $\mathbf{u}$ corresponding to global height, global velocity, attitude, and control input. We build the ReLU network using PyTorch [30]. Our ReLU network consists of four fully-connected hidden layers, with input and the output dimensions 12 and 3, respectively. We use spectral normalization (2.5) to constrain the Lipschitz constant of the DNN.

We compare the near-ground estimation accuracy our DNN model with existing 1D steady ground effect model [1], [3]:

$$T(n, z) = \frac{n^2}{1 - \mu(\frac{D}{8z})^2} c_T(n) = n^2 c_T(n_0) + \bar{f}_{a,z}, \tag{2.15}$$

where T is the thrust generated by propellers, n is the rotation speed, n_0 is the idle RPM, and μ depends on the number and the arrangement of propellers ($\mu = 1$ for a single propeller, but must be tuned for multiple propellers). Note that c_T is a function of n. Thus, we can derive $\bar{f}_{a,z}(n, z)$ from $T(n, z)$.

Fig. 2.2(a) shows the comparison between the estimated $\mathbf{f}_a$ from DNN and the theoretical ground effect model (2.15) at different z (assuming $T = mg$ when $z = \infty$). We can see that our DNN can achieve much better estimates than the theoretical ground effect model. We further investigate the trend of $\bar{f}_{a,z}$ with respect to the rotation speed n. Fig. 2.2(b) shows the learned $\hat{f}_{a,z}$ over the rotation speed n at a given height, in comparison with the C_T measured from the bench test. We observe that the increasing trend of the estimates $\hat{f}_{a,z}$ is consistent with bench test results for C_T.

To understand the benefits of SN, we compared $\hat{f}_{a,z}$ predicted by the DNNs trained both with and without SN as shown in Fig. 2.2(c). Note that v_z from -1 m/s to 1 m/s is covered in our training set, but -2 m/s to -1 m/s is not. We observe the following differences:

1. Ground effect: $\hat{f}_{a,z}$ increases as z decreases, which is also shown in Fig. 2.2(a).
2. Air drag: $\hat{f}_{a,z}$ increases as the drone goes down ($v_z < 0$) and it decreases as the drone goes up ($v_z > 0$).
3. Generalization: the spectral normalized DNN is much smoother and can also generalize to new input domains not contained in the training set.

In [18], the authors theoretically show that spectral normalization can provide tighter generalization guarantees on unseen data, which is consistent with our empirical

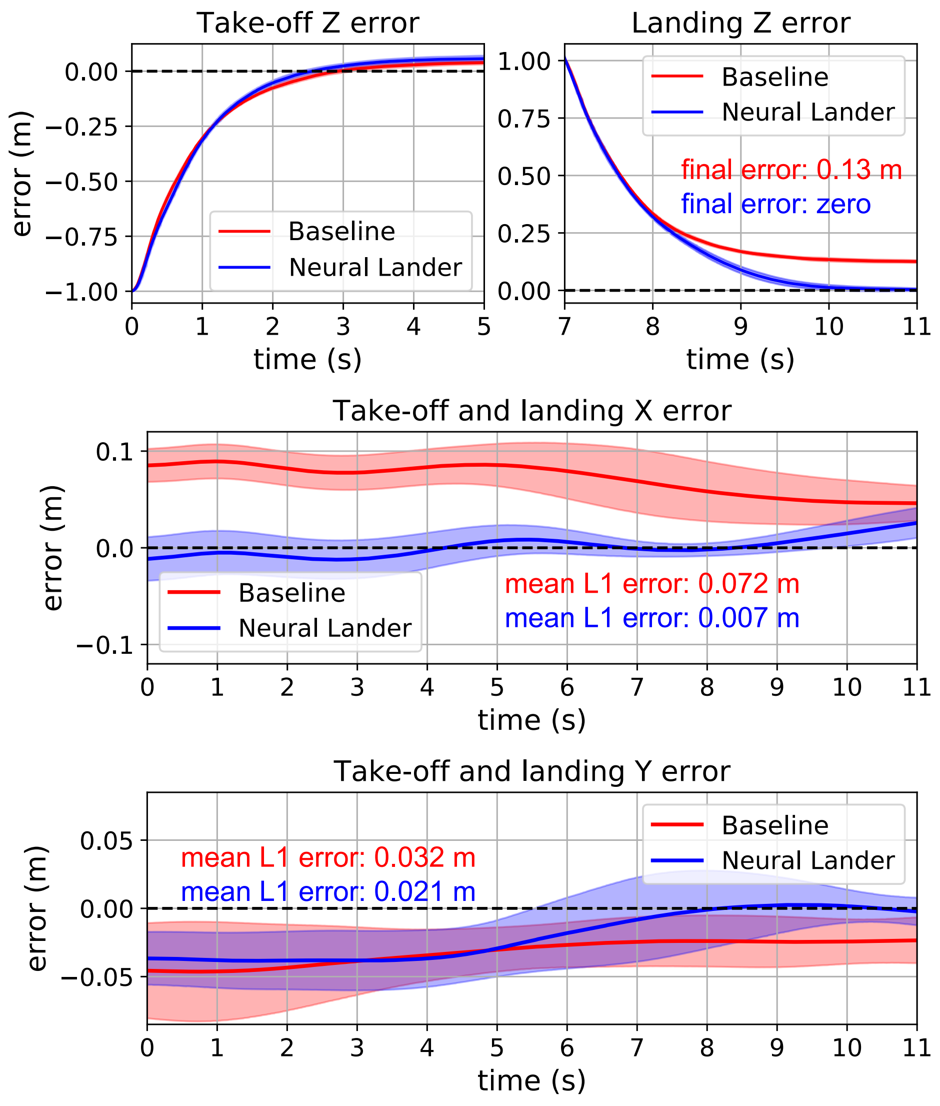


Figure 2.3: **Neural-network ground effect prediction.** Baseline Controller and *Neural-Lander* performance in take-off and landing. Means (solid curves) and standard deviations (shaded areas) of 10 trajectories.

observation. We will connect generalization theory more tightly with our robustness guarantees in the future.

Baseline Controller

We compared the *Neural-Lander* with a Baseline Nonlinear Tracking Controller. We implemented both a Baseline Controller similar to (2.7) and (2.8) with $\hat{\mathbf{f}}_a \equiv 0$, as well as an integral controller variation with $\mathbf{v}_r = \dot{\mathbf{p}}_d - 2\Lambda\tilde{\mathbf{p}} - \Lambda^2 \int_0^t \tilde{\mathbf{p}}(\tau)d\tau$. Though an integral gain can cancel steady-state error during set-point regulation, our flight results showed that the performance can be sensitive to the integral gain, especially

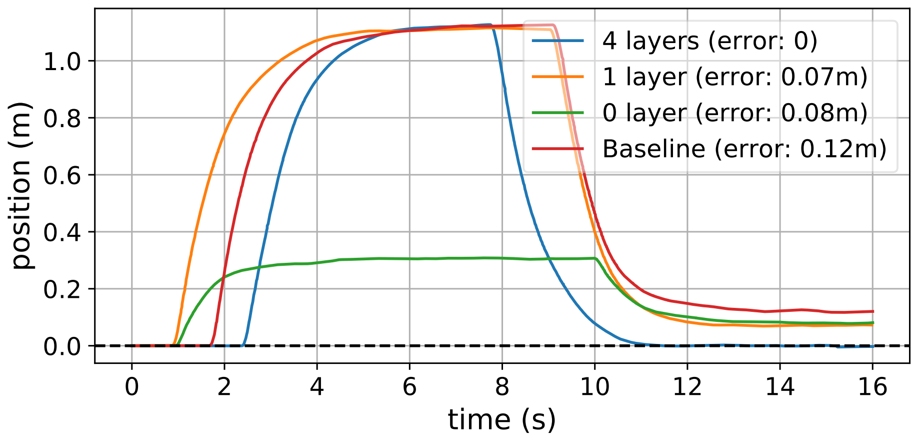


Figure 2.4: ***Neural-Lander* performance with difference numbers of hidden layers.** *Neural-Lander* performance in take-off and landing with different DNN capacities. 1 layer means $\hat{\mathbf{f}}_a = A\mathbf{x} + \mathbf{b}$; 0 layer means $\hat{\mathbf{f}}_a = \mathbf{b}$; Baseline means $\hat{\mathbf{f}}_a \equiv 0$.

during trajectory tracking. This can be seen in the demo video.[2]

Setpoint Regulation Performance

First, we tested the two controllers' performance in take-off/landing, by commanding position setpoint $\mathbf{p}_d$, from $(0, 0, 0)$, to $(0, 0, 1)$, then back to $(0, 0, 0)$, with $\dot{\mathbf{p}}_d \equiv 0$. From Fig. 2.3, we can conclude that there are two main benefits of our *Neural-Lander*. (a) *Neural-Lander* can control the drone to precisely and smoothly land on the ground surface while the Baseline Controller struggles to achieve 0 terminal height due to the ground effect. (b) *Neural-Lander* can mitigate drifts in $x - y$ plane, as it also learned about additional aerodynamics such as air drag.

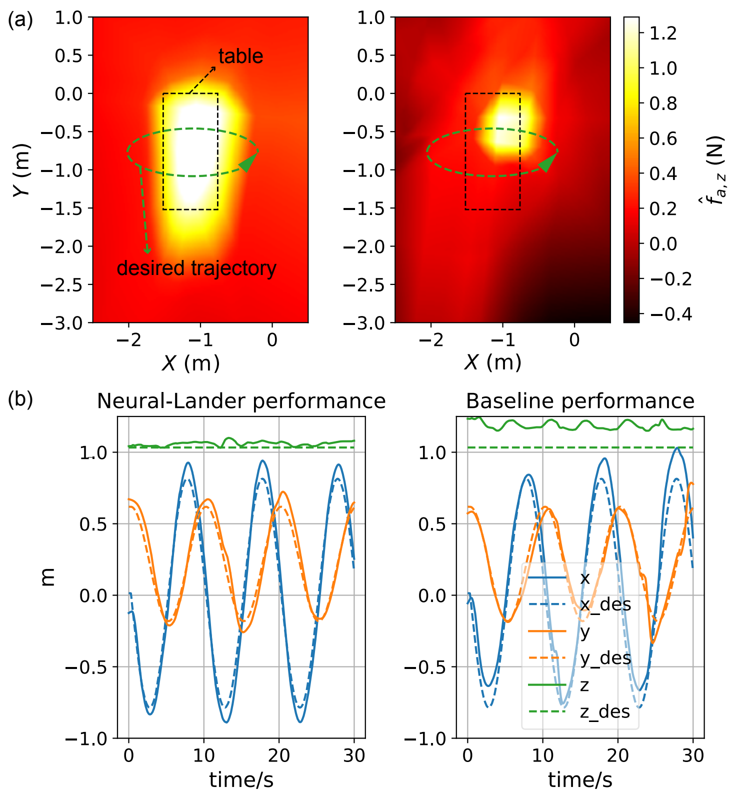
Second, we tested *Neural-Lander* performance with different DNN capacities. Fig. 2.4 shows that compared to the baseline ($\hat{\mathbf{f}}_a \equiv 0$), 1 layer model could decrease z error but it is not enough to land the drone. 0 layer model generated significant error during take-off.

In experiments, we observed the *Neural-Lander* without spectral normalization can even result in unexpected controller outputs leading to crash, which empirically implies the necessity of SN in training the DNN and designing the controller.

Trajectory Tracking Performance

To show that our algorithm can handle more complicated environments where physics-based modelling of dynamics would be substantially more difficult, we devise a task

[2]Demo videos: `https://youtu.be/FLLsG0S78ik`



	Neural-Lander	Baseline
Mean X error	**0.079m**	0.126m
Mean Y error	**0.052m**	0.061m
Mean Z error	**0.027m**	0.153m
Z variance	**0.014m**	0.026m

Figure 2.5: ***Neural-Lander* performance tracking a trajectory close to an obstacle.** (a) Heatmaps of learned $\hat{f}_{a,z}$ versus x and y, with other inputs fixed. (Left) ReLU network with spectral normalization. (Right) ReLU network without spectral normalization. (b) Tracking performance and statistics.

of tracking an elliptic trajectory very close to a table with a period of 10 seconds shown in Fig. 2.5. The trajectory is partially over the table with significant ground effects, and a sharp transition to free space at the edge of the table. We compared the performance of both *Neural-Lander* and Baseline Controller on this test.

In order to model the complex dynamics near the table, we manually flew the drone in the space close to the table to collect another data set. We trained a new ReLU DNN model with x-y positions as additional input features: $\hat{\mathbf{f}}_a(\mathbf{p}, \mathbf{v}, R, \mathbf{u})$. Similar to the setpoint experiment, the benefit of spectral normalization can be seen in Fig. 2.5(a), where only the spectrally-normalized DNN exhibits a clear table boundary.

Fig. 2.5(b) shows that *Neural-Lander* outperformed the Baseline Controller for tracking the desired position trajectory in all x, y, and z axes. Additionally, *Neural-Lander* showed a lower variance in height, even at the edge of the table, as the controller captured the changes in ground effects when the drone flew over the table.

In summary, the experimental results with multiple ground interaction scenarios show that much smaller tracking errors are obtained by *Neural-Lander*, which is essentially the nonlinear tracking controller with feedforward cancellation of a spectrally-normalized DNN.

2.7 Conclusions

In this paper, we present *Neural-Lander*, a deep-learning-based nonlinear controller with guaranteed stability for precise quadrotor landing. Compared to the Baseline Controller, *Neural-Lander* is able to significantly improve control performance. The main benefits are (1) our method can learn from coupled unsteady aerodynamics and vehicle dynamics to provide more accurate estimates than theoretical ground effect models, (2) our model can capture both the ground effect and other non-dominant aerodynamics and outperforms the conventional controller in all axes (x, y, and z), and (3) we provide rigorous theoretical analysis of our method and guarantee the stability of the controller, which also implies generalization to unseen domains.

Future work includes further generalization of the capabilities of *Neural-Lander* handling unseen state and disturbance domains, such as those generated by a wind fan array.

ACKNOWLEDGEMENT

The authors thank Joel Burdick and Daniel Pastor Moreno. The work is funded in part by Caltech's Center for Autonomous Systems and Technologies and Raytheon Company.

References

[1] I. C. Cheeseman and W. E. Bennett, "The effect of the ground on a helicopter rotor in forward flight," H.M. Stationery Office, 3021, 1957. [Online]. Available: `https://reports.aerade.cranfield.ac.uk/handle/1826.2/3590` (visited on 06/06/2023).

[2] K. Nonaka and H. Sugizaki, "Integral sliding mode altitude control for a small model helicopter with ground effect compensation," in *American Control Conference (ACC), 2011*, IEEE, 2011, pp. 202–207.

[3] L. Danjun, Z. Yan, S. Zongying, and L. Geng, “Autonomous landing of quadrotor based on ground effect modelling,” in *Control Conference (CCC), 2015 34th Chinese*, IEEE, 2015, pp. 5647–5652.

[4] F. Berkenkamp, A. P. Schoellig, and A. Krause, “Safe controller optimization for quadrotors with Gaussian processes,” in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 493–496. [Online]. Available: `https://arxiv.org/abs/1509.01066`.

[5] P. Abbeel, A. Coates, and A. Y. Ng, “Autonomous helicopter aerobatics through apprenticeship learning,” *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[6] A. Punjani and P. Abbeel, “Deep learning helicopter dynamics models,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 3223–3230. DOI: `10.1109/ICRA.2015.7139643`.

[7] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, “Learning quadrotor dynamics using neural network for flight control,” in *Decision and Control (CDC), 2016 IEEE 55th Conference on*, IEEE, 2016, pp. 4653–4660.

[8] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, “Deep neural networks for improved, impromptu trajectory tracking of quadrotors,” in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 5183–5189.

[9] S. Zhou, M. K. Helwa, and A. P. Schoellig, “Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking,” in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec. 2017, pp. 5201–5207. DOI: `10.1109/CDC.2017.8264430`.

[10] C. Sánchez-Sánchez and D. Izzo, “Real-time optimal control via deep neural networks: Study on landing problems,” *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 5, pp. 1122–1135, 2018.

[11] S. Balakrishnan and R. Weil, “Neurocontrol: A literature survey,” *Mathematical and Computer Modelling*, vol. 23, no. 1-2, pp. 101–117, 1996.

[12] M. T. Frye and R. S. Provence, “Direct inverse control using an artificial neural network for the autonomous hover of a helicopter,” in *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, IEEE, 2014, pp. 4121–4122.

[13] H. Suprijono and B. Kusumoputro, “Direct inverse control based on neural network for unmanned small helicopter attitude and altitude control,” *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 2-2, pp. 99–102, 2017.

[14] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” in *Proc. of Neural Information Processing Systems (NIPS)*, 2017. [Online]. Available: `https://arxiv.org/abs/1705.08551`.

[15] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral normalization for generative adversarial networks,” Feb. 16, 2018. arXiv: `1802.05957 [cs, stat]`. [Online]. Available: `http://arxiv.org/abs/1802.05957` (visited on 09/02/2021).

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[17] T. Salimans and D. P. Kingma, “Weight normalization: A simple reparameterization to accelerate training of deep neural networks,” in *Advances in Neural Information Processing Systems*, 2016, pp. 901–909.

[18] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky, “Spectrally-normalized margin bounds for neural networks,” in *Advances in Neural Information Processing Systems*, 2017, pp. 6240–6249.

[19] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, N.J: Prentice Hall, 1991, 459 pp., ISBN: 978-0-13-040890-7.

[20] S. Bandyopadhyay, S.-J. Chung, and F. Y. Hadaegh, “Nonlinear attitude control of spacecraft with a large captured object,” *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 4, pp. 754–769, Apr. 2016, ISSN: 0731-5090, 1533-3884. DOI: `10.2514/1.G001341`. [Online]. Available: `https://arc.aiaa.org/doi/10.2514/1.G001341` (visited on 03/09/2022).

[21] X. Shi, K. Kim, S. Rahili, and S.-J. Chung, “Nonlinear control of autonomous flying cars with wings and distributed electric propulsion,” in *2018 IEEE Conference on Decision and Control (CDC)*, Miami Beach, FL: IEEE, Dec. 2018, pp. 5326–5333, ISBN: 978-1-5386-1395-5. DOI: `10.1109/CDC.2018.8619578`. [Online]. Available: `https://ieeexplore.ieee.org/document/8619578/` (visited on 08/25/2020).

[22] D. Morgan, G. P. Subramanian, S.-J. Chung, and F. Y. Hadaegh, “Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming,” *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1261–1285, Sep. 1, 2016, ISSN: 0278-3649. DOI: `10.1177/0278364916632065`. [Online]. Available: `https://doi.org/10.1177/0278364916632065` (visited on 06/06/2023).

[23] H. K. Khalil, *Nonlinear Systems, 3rd Edition*. Prentice Hall, 2002. [Online]. Available: `https://www.pearson.com/content/one-dot-com/one-dot-com/us/en/higher-education/program.html` (visited on 09/02/2021).

[24] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning requires rethinking generalization,” *arXiv preprint arXiv:1611.03530*, 2016.

[25] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[26] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro, “A pac-bayesian approach to spectrally-normalized margin bounds for neural networks,” *arXiv preprint arXiv:1707.09564*, 2017.

[27] G. K. Dziugaite and D. M. Roy, “Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data,” *arXiv preprint arXiv:1703.11008*, 2017.

[28] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro, “Exploring generalization in deep learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5947–5956.

[29] S.-J. Chung, S. Bandyopadhyay, I. Chang, and F. Y. Hadaegh, “Phase synchronization control of complex networks of Lagrangian systems on adaptive digraphs,” *Automatica*, vol. 49, no. 5, pp. 1148–1161, 2013.

[30] A. Paszke, S. Gross, S. Chintala, *et al.*, “Automatic differentiation in PyTorch,” presented at the Advances in Neural Information Processing Systems, 2017.

Chapter 3

NEURAL-FLY: OFFLINE LEARNING FOR RAPID AND ROBUST ONLINE ADAPTATION

3.1 Introduction

The commoditization of uninhabited aerial vehicles (UAVs) requires that the control of these vehicles become more precise and agile. For example, drone delivery requires transporting goods to a narrow target area in various weather conditions; drone rescue and search require entering and searching collapsed buildings with little space; urban air mobility needs a flying car to follow a planned trajectory closely to avoid collision in the presence of strong unpredictable winds.

Unmodeled and often complex aerodynamics are among the most notable challenges to precise flight control. Flying in windy environments (as shown in Fig. 3.1) introduces even more complexity because of the unsteady aerodynamic interactions between the drone, the induced airflow, and the wind (see Fig. 3.1(F) for a smoke visualization). These unsteady and nonlinear aerodynamic effects substantially degrade the performance of conventional UAV control methods that neglect to account for them in the control design. Prior approaches partially capture these effects with simple linear or quadratic air drag models, which limit the tracking performance in agile flight and cannot be extended to external wind conditions [1], [2]. Although more complex aerodynamic models can be derived from computational fluid dynamics [3], such modelling is often computationally expensive, and is limited to steady non-dynamic wind conditions. Adaptive control addresses this problem by estimating linear parametric uncertainty in the dynamical model in real time to improve tracking performance. Recent state-of-the-art in quadrotor flight control has used adaptive control methods that directly estimate the unknown aerodynamic force without assuming the structure of the underlying physics, but relying on high-frequency and low-latency control [4]–[7]. In parallel, there has been increased interest in data-driven modeling of aerodynamics (e.g., [8]–[11]), however existing approaches cannot effectively adapt in changing or unknown environments such as time-varying wind conditions.

In this article, we present a data-driven approach called Neural-Fly, which is a deep-learning-based trajectory tracking controller that learns to quickly adapt to rapidly-

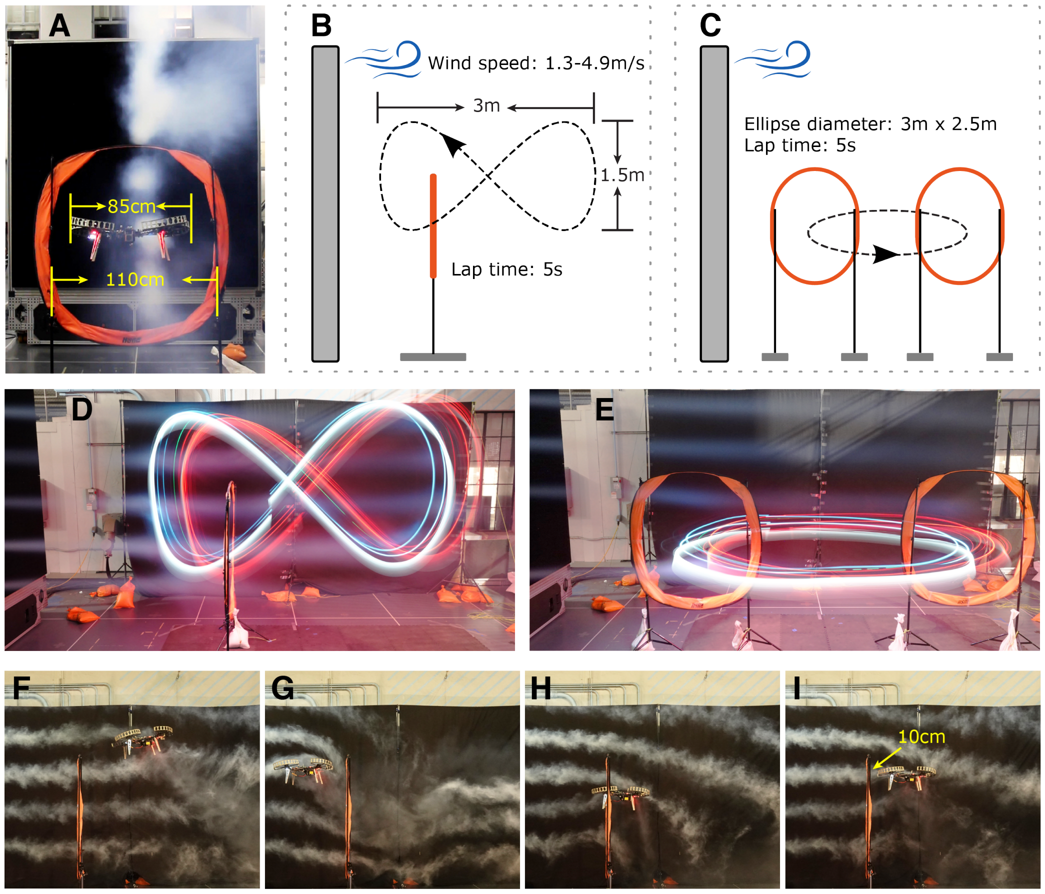


Figure 3.1: **Agile flight through narrow gates.** (**A**) Caltech Real Weather Wind Tunnel system, the quadrotor UAV, and the gate. In our flight tests, the UAV follows an agile trajectory through narrow gates, which are slightly wider than the UAV itself, under challenging wind conditions. (**B-C**) Trajectories used for the gate tests. In (B), the UAV follows a figure-8 through one gate, with wind speed 3.1 m/s or time-varying wind condition. In (C), the UAV follows an ellipse in the horizontal plane through two gates, with wind speed 3.1 m/s. (**D-E**) Long-exposure photos (with an exposure time of 5 s) showing one lap in two tasks. (**F-I**) High-speed photos (with a shutter speed of 1/200s) showing the moment the UAV passed through the gate and the interaction between the UAV and the wind.

changing wind conditions. Our method, depicted in Fig. 3.2, advances and offers insights into both adaptive flight control and deep-learning-based robot control. Our experimental demonstrates that Neural-Fly achieves centimeter-level position-error tracking of an agile and challenging trajectory in dynamic wind conditions on a standard UAV.

Our method has two main components: an offline learning phase and an online adaptive control phase used as real-time online learning. For the offline learning phase, we have developed Domain Adversarially Invariant Meta-Learning (DAIML) that learns a wind-condition-independent deep neural network (DNN) representation of the aerodynamics in a data-efficient manner. The output of the DNN is treated as a set of basis functions that represent the aerodynamic effects. This representation is adapted to different wind conditions by updating a set of linear coefficients that mix the output of the DNN. DAIML is data efficient and uses only 12 total minutes of flight data in just 6 different wind conditions to train the DNN. DAIML incorporates several key features which not only improve the data efficiency but also are informed by the downstream online adaptive control phase. In particular, DAIML uses spectral normalization [8], [12] to control the Lipschitz property of the DNN to improve generalization to unseen data and provide closed-loop stability and robustness guarantees. DAIML also uses a discriminative network, which ensures that the learned representation is wind-invariant and that the wind-dependent information is only contained in the linear coefficients that are adapted in the online control phase.

For the online adaptive control phase, we have developed a regularized composite adaptive control law, which we derived from a fundamental understanding of how the learned representation interacts with the closed-loop control system and which we support with rigorous theory. The adaptation law updates the wind-dependent linear coefficients using a composite of the position tracking error term and the aerodynamic force prediction error term. Such a principled approach effectively guarantees stable and fast adaptation to any wind condition and robustness against imperfect learning. Although this adaptive control law could be used with a number of learned models, the speed of adaptation is further aided by the concise representation learned from DAIML.

Using Neural-Fly, we report an average improvement of 66 % over a nonlinear tracking controller, 42 % over an $\mathcal{L}_1$ adaptive controller, and 35 % over an Incremental Nonlinear Dynamics Inversion (INDI) controller. These results are all accomplished using standard quadrotor UAV hardware, while running the PX4's default regulation

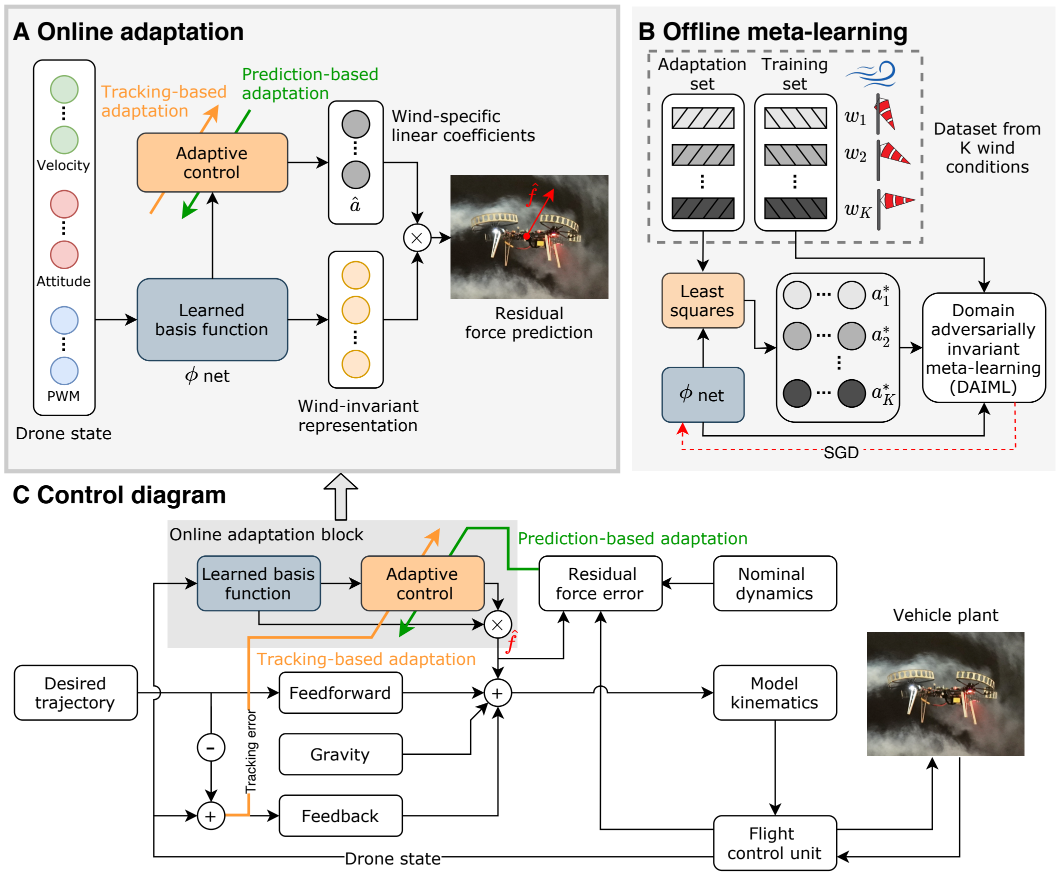


Figure 3.2: **Offline meta-learning and online adaptive control design.** (**A**) The online adaptation block in our adaptive controller. Our controller leverages the meta-trained basis function ϕ, which is a wind-invariant representation of the aerodynamic effects, and uses composite adaptation (that is, including tracking-error-based and prediction-error-based adaptation) to update wind-specific linear weights $\hat{a}$. The output of this block is the wind-effect force estimate, $\hat{f} = \phi\hat{a}$. (**B**) The illustration of our meta-learning algorithm DAIML. We collected data from wind conditions $\{w_1, \cdots, w_K\}$ and applied Algorithm 1 to train the ϕ net. (**C**) The diagram of our control method, where the gray part corresponds to (A). Interpreting the learned block as an aerodynamic force allows it to be incorporated into the feedback control easily.

attitude control. Our tracking performance is competitive even compared to related work without external wind disturbances and with more complex hardware (for example, [4] requires a 10-time higher control frequency and onboard optical sensors for direct motor speed feedback). We also compare Neural-Fly with two variants of our method: Neural-Fly-Transfer, which uses a learned representation trained on data from a different drone, and Neural-Fly-Constant, which only uses our adaptive control law with a trivial non-learning basis. Neural-Fly-Transfer demonstrates that our method is robust to changes in vehicle configuration and model mismatch. Neural-Fly-Constant, $\mathcal{L}_1$, and INDI all directly adapt to the unknown dynamics without assuming the structure of the underlying physics, and they have similar performance. Furthermore, we demonstrate that our method enables a new set of capabilities that allow the UAV to fly through low-clearance gates following agile trajectories in gusty wind conditions (Fig. 3.1).

Related Work for Precise Quadrotor Control

Typical quadrotor control consists of a cascaded or hierarchical control structure which separates the design of the position controller, attitude controller, and thrust mixer (allocation). Commonly-used off-the-shelf controllers, such as PX4, design each of these loops as proportional-integral-derivative (PID) regulation controllers [13]. The control performance can be substantially improved by designing each layer of the cascaded controller as a tracking controller using the concept of differential flatness [14], or, as has recently been popular, using a single optimization-based controller such as model predictive control (MPC) to directly compute motor speed commands from desired trajectories. State-of-the-art tracking performance relies on MPC with fast adaptive inner loops to correct for modeling errors [4], [7], however, this approach requires full custom flight controllers. In contrast, our method is designed to be integrated with a typical PX4 flight controller, yet it achieves state-of-the-art flight performance in wind.

Prior work on agile quadrotor control has achieved impressive results by considering aerodynamics [2], [4], [7], [11]. However, those approaches require specialized onboard hardware [4], full custom flight control stacks [4], [7], or cannot adapt to external wind disturbances [2], [11]. For example, state-of-the-art tracking performance has been demonstrated using incremental nonlinear dynamics inversion to estimate aerodynamic disturbance forces, with a root-mean-square tracking error of 6.6 cm and drone ground speeds up to 12.9 m/s [4]. However, [4] relies on high-frequency control updates (500 Hz) and direct motor speed feedback using optical encoders to rapidly estimate external disturbances. Both are challenging to deploy on standard systems. [7] simplifies the hardware setup and does not require optical motor speed sensors and has demonstrated state-of-the-art tracking performance. However, [7] relies on a high-rate $\mathcal{L}_1$ adaptive controller inside a model predictive controller and uses a racing drone with a fully customized control stack. [11] leverages an aerodynamic model learned offline and represented as Gaussian Processes. However, [11] cannot adapt to unknown or changing wind conditions and provides no theoretical guarantees. Another recent work focuses on deriving simplified rotor-drag models that are differentially flat [2]. However, [2] focuses on horizontal, $xy-$plane trajectories at ground speeds of $4\,\mathrm{m/s}$ without external wind, where the thrust is more constant than ours, achieves ~6 cm tracking error [2], uses an attitude controller running at 4000 Hz, and is not extensible to faster flights as pointed out in [11].

Relation between Neural-Fly and Conventional Adaptive Control

Adaptive control theory has been extensively studied for online control and identification problems with parametric uncertainty, for example, unknown linear coefficients for mixing known basis functions [15]–[20]. There are three common aspects of adaptive control which must be addressed carefully in any well-designed system and which we address in Neural-Fly: designing suitable basis functions for online adaptation, stability of the closed-loop system, and persistence of excitation, which is a property related to robustness against disturbances. These challenges arise due to the coupling between the unknown underlying dynamics and the online adaptation. This coupling precludes naive combinations of online learning and control. For example, gradient-based parameter adaptation has well-known stability and robustness issues as discussed in [15].

The basis functions play a crucial role in the performance of adaptive control, but designing or selecting proper basis functions might be challenging. A good set of basis functions should reflect important features of the underlying physics. In practice, basis functions are often designed using physics-informed modeling of the system, such as the nonlinear aerodynamic modeling in [21]. However, physics-informed modeling requires a tremendous amount of prior knowledge and human labor, and is often still inaccurate. Another approach is to use random features as the basis set, such as random Fourier features [22], [23], which can model all possible underlying physics as long as the number of features is large enough. However, the high-dimensional feature space is not optimal for a specific system because many of the features might be redundant or irrelevant. Such suboptimality and redundancy not only increase the computational burden but also slow down the convergence speed of the adaptation process.

Given a set of basis functions, naive adaptive control designs may cause instability and fragility in the closed-loop system, due to the nontrivial coupling between the adapted model and the system dynamics. In particular, asymptotically stable adaptive control cannot guarantee robustness against disturbances and so exponential stability is desired. Even so, often, existing adaptive control methods only guarantee exponential stability when the desired trajectory is persistently exciting, by which information about all the coefficients (including irrelevant ones) is constantly provided at the required spatial and time scales. In practice, persistent excitation requires either a succinct set of basis functions or perturbing the desired trajectory, which compromises tracking performance.

Recent multirotor flight control methods, including INDI [4] and $\mathcal{L}_1$ adaptive control, presented in [5] and demonstrated inside a model predictive control loop in [7], achieve good results by abandoning complex basis functions. Instead, these methods directly estimate the aerodynamic residual force vector. The residual force is observable, thus, these methods bypass the challenge of designing good basis functions and the associated stability and persistent excitation issues. However, these methods suffer from lag in estimating the residual force and encounter the filter design performance trade of reduced lag versus amplified noise. Neural-Fly-Constant only uses Neural-Fly's composite adaptation law to estimate the residual force, and therefore, Neural-Fly-Constant also falls into this class of adaptive control structures. The results of this article demonstrate that the inherent estimation lag in these existing methods limits performance on agile trajectories and in strong wind conditions.

Neural-Fly solves the aforementioned issues of basis function design and adaptive control stability, using newly developed methods for meta-learning and composite adaptation that can be seamlessly integrated together. Neural-Fly uses DAIML and flight data to learn an effective and compact set of basis functions, represented as a DNN. The regularized composite adaptation law uses the learned basis functions to quickly respond to wind conditions. Neural-Fly enjoys fast adaptation because of the conciseness of the feature space, and it guarantees closed-loop exponential stability and robustness without assuming persistent excitation.

Related to Neural-Fly, neural network-based adaptive control has been researched extensively, but by and large was limited to shallow or single-layer neural networks without pretraining. Some early works focus on shallow or single-layer neural networks with unknown parameters which are adapted online [19], [24]–[27]. A recent work applies this idea to perform an impressive quadrotor flip [28]. However, the existing neural network-based adaptive control work does not employ multi-layer DNNs, and lacks a principled and efficient mechanism to pretrain the neural network before deployment. Instead of using shallow neural networks, recent trends in machine learning highly rely on DNNs due to their representation power [29]. In this work, we leverage modern deep-learning advances to pretrain a DNN which represents the underlying physics compactly and effectively.

Related Work in Multi-environment Deep-Learning for Robot Control

Recently, researchers have been addressing the data and computation requirements for DNNs to help the field progress towards the fast online-learning paradigm. In turn, this progress has been enabling adaptable DNN-based control in dynamic environments. The most popular learning scheme in dynamic environments is meta-learning, or "learning-to-learn," which aims to learn an efficient model from data across different tasks or environments [30]–[32]. The learned model, typically represented as a DNN, ideally should be capable of rapid adaptation to a new task or an unseen environment given limited data. For robotic applications, meta-learning has shown great potential for enabling autonomy in highly-dynamic environments. For example, it has enabled quick adaptation against unseen terrain or slopes for legged robots [33], [34], changing suspended payload for drones [35], and unknown operating conditions for wheeled robots [36].

In general, 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 different environments, such that the model contains shared knowledge or features across all environment, for example, learning aerodynamic features shared by all wind conditions. In the online adaptation phase, the goal is to adapt the offline-learned model, given limited online data from a new environment or a new task, for example, fine-tuning the aerodynamic features in a specific wind condition.

There are two ways that the offline-learned model can be adapted. In the first class, the adaptation phase adapts the whole neural network model, typically using one or more gradient descent steps [30], [33], [35], [37]. However, due to the notoriously data-hungry and high-dimensional nature of neural networks, for real-world robots it is still impossible to run such adaptation on-board as fast as the feedback control loop (e.g., ~100Hz for quadrotor). Furthermore, adapting the whole neural network often lacks explainability and robustness and could generate unpredictable outputs that make the closed-loop unstable.

In the second class (including Neural-Fly), the online adaptation only adapts a relatively small part of the learned model, for example, the last layer of the neural network [36], [38]–[40]. The intuition is that, different environments share a common representation (e.g., the wind-invariant representation in Fig. 3.2(A)), and the environment-specific part is in a low-dimensional space (e.g., the wind-specific linear weight in Fig. 3.2(A)), which enables the real-time adaptation as fast as the control loop. In particular, the idea of integrating meta-learning with adaptive control is first presented in our prior work [38], later followed by [39]. However, the representation learned in [38] is ineffective and the tracking performance in [38] is similar as the baselines; [39] focuses on a planar and fully-actuated rotorcraft simulation without experiment validation and there is no stability or robustness analysis. Neural-Fly instead learns an effective representation using our meta-learning algorithm called DAIML, demonstrates state-of-the-art tracking performance on real drones, and achieves non-trivial stability and robustness guarantees.

Another popular deep-learning approach for control in dynamic environments is robust policy learning via domain randomization [41]–[43]. The key idea is to train the policy with random physical parameters such that the controller is robust to a range of conditions. For example, the quadrupedal locomotion controller in [41] retains its robustness over challenging natural terrains. However, robust policy learning optimizes average performance under a broad range of conditions rather than achieving precise control by adapting to specific environments.

3.2 Results

In this section, we first discuss the experimental platform for data collection and experiments. Second, we discuss the key conceptual reasoning behind our combined method of our meta-learning algorithm, called DAIML, and our composite adaptive controller with stability guarantees. Third, we discuss several experiments to quantitatively compare the closed-loop trajectory-tracking performance of our methods to a nonlinear baseline method and two state-of-the-art adaptive flight control methods, and we observe our methods reduce the average tracking error substantially. In order to demonstrate the new capabilities brought by our methods, we present agile flight results in gusty winds, where the UAV must quickly fly through narrow gates that are only slightly wider than the vehicle. Finally, we show our methods are also applicable in outdoor agile tracking tasks without external motion capture systems.

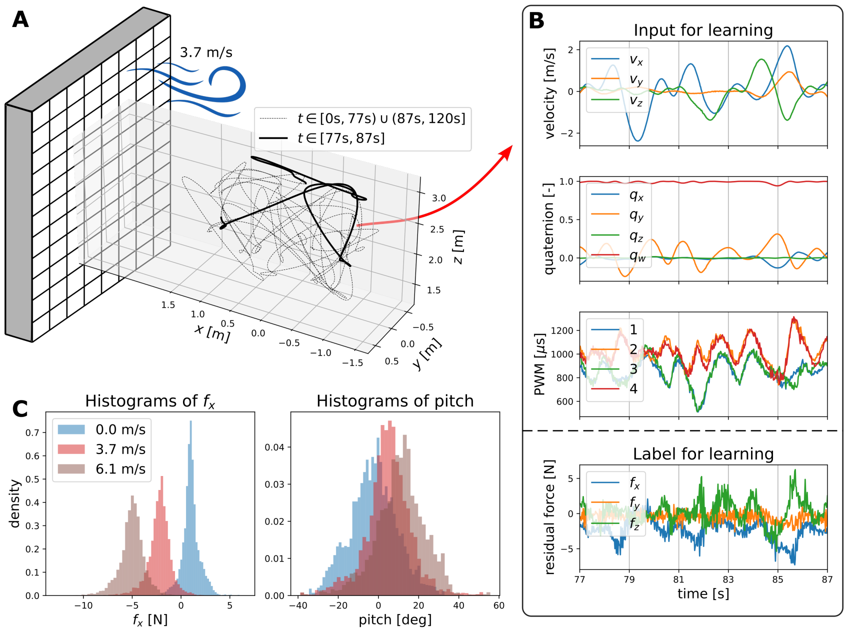


Figure 3.3: **Training data collection.** (**A**) The xyz position along a two-minute randomized trajectory for data collection with wind speed 8.3 km/h (3.7 m/s), in the Caltech Real Weather Wind Tunnel. (**B**) A typical 10-second trajectory of the inputs (velocity, attitude quaternion, and motor speed PWM command) and label (offline calculation of aerodynamic residual force) for our learning model, corresponding to the highlighted part in (A). (**C**) Histograms showing data distributions in different wind conditions. (**C**) **Left:** distributions of the x-component of the wind-effect force, f_x. This shows that the aerodynamic effect changes as the wind varies. (**C**) **Right:** distributions of the pitch, a component of the state used as an input to the learning model. This shows that the shift in wind conditions causes a distribution shift in the input.

Experimental Platform

All of our experiments are conducted at Caltech's Center for Autonomous Systems and Technologies (CAST). The experimental setup consists of an OptiTrack motion capture system with 12 infrared cameras for localization streaming position measurements at 50 Hz, a Wi-Fi router for communication, the Caltech Real Weather Wind Tunnel for generating dynamic wind conditions, and a custom-built quadrotor UAV. The Real Weather Wind Tunnel is composed of 1296 individually controlled fans and can generate uniform wind speeds of up to 43.6 km/h in its 3x3x5m test section. For outdoor flight, the drone is also equipped with a Global Positioning

System (GPS) module and an external antenna. We now discuss the design of the UAV and the wind condition in detail.

UAV Design We built a quadrotor UAV for our primary data collection and all experiments, shown in Fig. 3.1(A). The quadrotor weighs 2.6 kg with a thrust to weight ratio of 2.2. The UAV is equipped with a Pixhawk flight controller running PX4, an open-source commonly used drone autopilot platform [13]. The UAV incorporates a Raspberry Pi 4 onboard computer running a Linux operating system, which performs real-time computation and adaptive control and interfaces with the flight controller through MAVROS, an open-source set of communication drivers for UAVs. State estimation is performed using the built-in PX4 Extended Kalman Filter (EKF), which fuses inertial measurement unit (IMU) data with global position estimates from OptiTrack motion capture system (or the GPS module for outdoor flight tasks). The UAV platform features a wide-X configuration, measuring 85 cm in width, 75 cm in length, and 93 cm diagonally, and tilted motors for improved yaw authority. This general hardware setup is standard and similar to many quadrotors. We refer to the supplementary materials (4.1) for further configuration details.

We implemented our control algorithm and the baseline control methods in the position control loop in Python, and run it on the onboard Linux computer at 50 Hz. The PX4 was set to the offboard flight mode and received thrust and attitude commands from the position control loop. The built-in PX4 multicopter attitude controller was then executed at the default rate, which is a linear PID regulation controller on the quaternion error. The online inference of the learned representation is also in Python via PyTorch, which is an open source deep-learning framework.

To study the generalizability and robustness of our approach, we also use an Intel Aero Ready to Fly drone for data collection. This dataset is used to train a representation of the wind effects on the Intel Aero drone, which we test on our custom UAV. The Intel Aero drone (weighing 1.4 kg) has a symmetric X configuration, 52 cm in width and 52 cm in length, without tilted motors (see the supplementary materials for further details).

Wind Condition Design To generate dynamic and diverse wind conditions for the data collection and experiments, we leverage the state-of-the-art Caltech Real Weather Wind Tunnel system (Fig. 3.1(A)). The wind tunnel is a 3 m by 3 m array of 1296 independently controllable fans capable of generating wind conditions up to

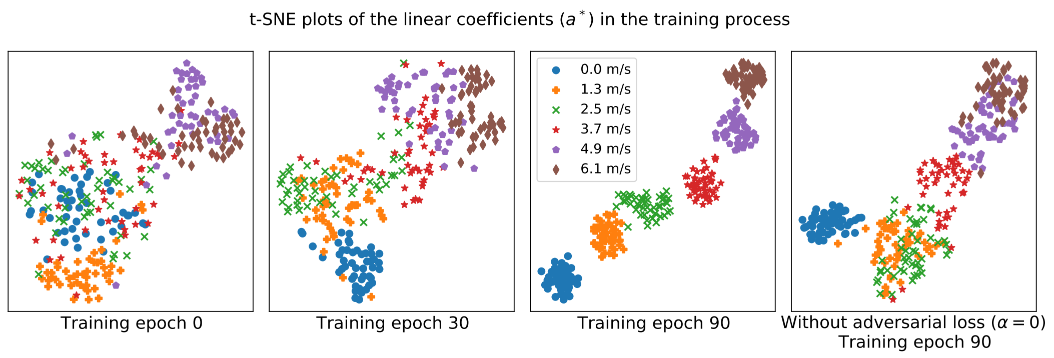


Figure 3.4: **t-SNE plots showing the evolution of the linear weights (a^*) during the training process.** As the number of training epochs increases, the distribution of a^* becomes more clustered with similar wind speed clusters near each other. The clustering also has a physical meaning: after training convergence, the right top part corresponds to a higher wind speed. This suggests that DAIML successfully learned a basis function ϕ shared by all wind conditions, and the wind-dependent information is contained in the linear weights. Compared to the case without the adversarial regularization term (using $\alpha = 0$ in Algorithm 1), the learned result using our algorithm is also more explainable, in the sense that the linear coefficients in different conditions are more disentangled.

43.6 km/h. The distributed fans are controlled in real-time by a Python-based Application Programming Interface (API). For data collection and flight experiments, we designed two types of wind conditions. For the first type, each fan has uniform and constant wind speed between 0 km/h and 43.6 km/h (12.1 m/s). The second type of wind follows a sinusoidal function in time, e.g., $30.6 + 8.6\sin(t)$ km/h. Note that the training data only covers constant wind speeds up to 6.1 m/s. To visualize the wind, we use 5 smoke generators to indicate the direction and intensity of the wind condition (see examples in Fig. 3.1 and Video 1).

Offline Learning and Online Adaptive Control Development

Data Collection and Meta-Learning using DAIML To learn an effective representation of the aerodynamic effects, we have a custom-built drone follow a randomized trajectory for 2 minutes each in six different static wind conditions, with speeds ranging from 0 km/h to 22.0 km/h. However, in experiments we used wind speeds up to 43.6 km/h (12.1 m/s) to study how our methods extrapolate to unseen wind conditions (e.g., Fig. 3.6). The data is collected at 50 Hz with a total of 36,000 data points. Figure 3.3(A) shows the data collection process, and Fig. 3.3(B) shows the inputs and labels of the training data, under one wind condition of 13.3 km/h (3.7 m/s). Figure 3.3(C) shows the distributions of input data (pitch) and label data ($x-$component of the aerodynamic force) in different wind conditions. Clearly, a shift in wind conditions causes distribution shifts in both input domain and label domain, which motivates the algorithm design of DAIML. The same data collection process is repeated on the Intel Aero drone, to study whether the learned representation can generalize to a different drone.

On the collected datasets for both our custom drone and the Intel Aero drone, we apply the DAIML algorithm to learn two representations ϕ of the wind effects. The learning process is done offline on a normal desktop computer, and depicted in Fig. 3.2(B). Figure 3.4 shows the evolution of the linear coefficients (a^*) during the learning process, where DAIML learns a representation of the aerodynamic effects shared by all wind conditions, and the linear coefficient contains the wind-specific information. Moreover, the learned representation is explainable in the sense that the linear coefficients in different wind conditions are well disentangled (see Fig. 3.4). We refer to the "Materials and Methods" section for more details.

Baselines and the Variants of Our Method We briefly introduce three variants of our method and the three baseline methods considered (details are provided in the "Materials and Methods" section). Each of the controllers is implemented in the position control loop and outputs a force command. The force command is fed into a kinematics block to determine a corresponding attitude and thrust, similar to [14], which is sent to the PX4 flight controller. The three baselines include: globally exponentially-stabilizing nonlinear tracking controller for quadrotor control [8], [44], [45], incremental nonlinear dynamics inversion (INDI) linear acceleration control [4], and $\mathcal{L}_1$ adaptive control [5], [7]. The primary difference between these baseline methods and Neural-Fly is how the controller compensates for the unmodeled residual force (that is, each baseline method has the same control structure, in Fig. 3.2(C), except for the estimation of the $\hat{f}$). In the case of the nonlinear baseline controller an integral term accumulates error to correct for the modeling error. The integral gain is limited by the stability of the interaction with the position and velocity error feedback leading to slow model correction. In contrast, both INDI and $\mathcal{L}_1$ decouple the adaptation rate from the PD gains, which allow for fast adaptation. Instead, these methods are limited by more fundamental design factors, such as system delay, measurement noise, and controller rate.

Our method is illustrated in Fig. 3.2(A,C) and replaces the integral feedback term with an adapted learning term. The deployment of our approach depends on the learned representation function ϕ, and our primary method and two variants consider a different choice of ϕ. Neural-Fly is our primary method using a representation learned from the dataset collected by the custom-built drone, which is the same drone used in experiments. Neural-Fly-Transfer uses the Neural-Fly algorithm where the representation is trained using the dataset collected by the aforementioned Intel Aero drone. Neural-Fly-Constant uses the online adaptation algorithm from Neural-Fly, but the representation is an artificially designed constant mapping. Neural-Fly-Transfer is included to show the generalizability and robustness of our approach with drone transfer, i.e., using a different drone in experiments than data collection. Finally, Neural-Fly-Constant demonstrates the benefit of using a better representation learned from the proposed meta-learning method DAIML. Note that Neural-Fly-Constant is a composite adaptation form of a Kalman-filter disturbance observer, that is a Kalman-filter augmented with a tracking error update term.

Trajectory Tracking Performance

We quantitatively compare the performance of the aforementioned control methods when the UAV follows a 2.5 m wide, 1.5 m tall figure-8 trajectory with a lap time of 6.28 s under constant, uniform wind speeds of 0 km/h, 15.1 km/h (4.2 m/s), 30.6 km/h (8.5 m/s), and 43.6 km/h (12.1 m/s) and under time-varying wind speeds of $30.6 + 8.6\sin(t)$ km/h ($8.5 + 2.4\sin(t)$ m/s).

The flight trajectory for each of the experiments is shown in Fig. 3.5, which includes a warm-up lap and six 6.28 s laps. The nonlinear baseline integral term compensates for the mean model error within the first lap. As the wind speed increases, the aerodynamic force variation becomes larger, and we notice a substantial performance degradation. INDI and $\mathcal{L}_1$ both improve over the nonlinear baseline, but INDI is more robust than $\mathcal{L}_1$ at high wind speeds. Neural-Fly-Constant outperforms INDI except during the two most challenging tasks: 43.6 km/h and sinusoidal wind speeds. The learning-based methods, Neural-Fly and Neural-Fly-Transfer, outperform all other methods in all tests. Neural-Fly outperforms Neural-Fly-Transfer slightly, which is because the learned model was trained on data from the same drone and thus better matches the dynamics of the vehicle.

In Table 3.1, we tabulate the root-mean-square position error and mean position error values over the six laps for each experiment. Figure 3.6 shows how the mean tracking error changes for each controller as the wind speed increases, and includes the standard deviation for the mean lap position error. In all cases, Neural-Fly and

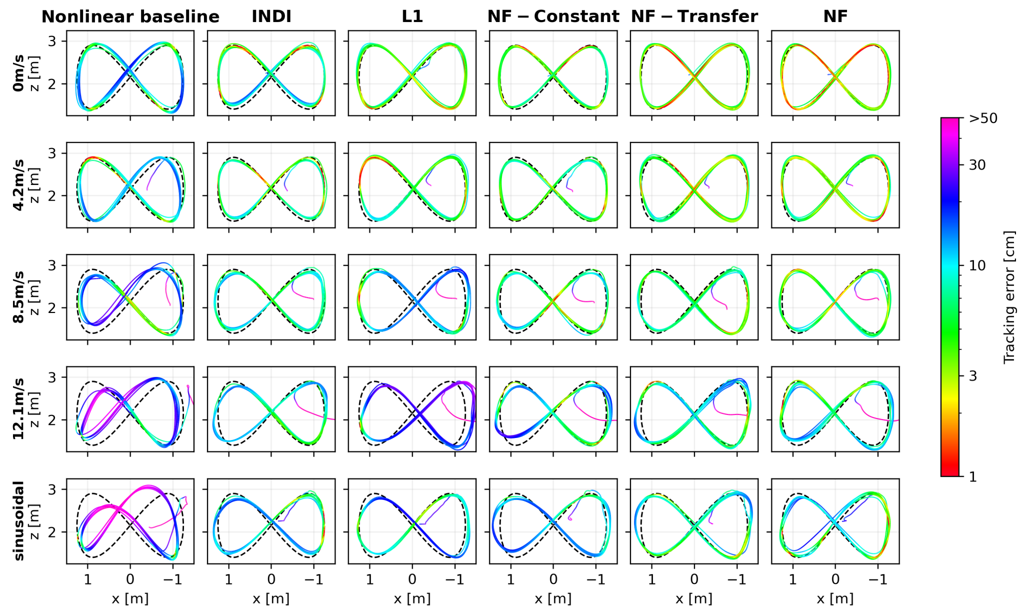


Figure 3.5: **Depiction of the trajectory tracking performance of each controller in several wind conditions.** The baseline nonlinear controller can track the trajectory well, however, the performance substantially degrades at higher wind speeds. INDI, $\mathcal{L}_1$, and Neural-Fly-Constant have similar performance and improve over the nonlinear baseline by estimating the aerodynamic disturbance force quickly. Neural-Fly and Neural-Fly-Transfer use a learned model of the aerodynamic effects and adapt the model in real time to achieve lower tracking error than the other methods.

Neural-Fly-Transfer outperform the state-of-the-art baseline methods, including the 30.6 km/h, the 43.6 km/h, and the sinusoidal wind speeds all of which exceed the wind speed in the training data. All of these results presents a clear trend: adaptive control substantially outperforms the nonlinear baseline which relies on integral-control, and learning markedly improves adaptive control.

Agile Flight Through Narrow Gates

Precise flight control in dynamic and strong wind conditions has many applications, such as rescue and search, delivery, and transportation. In this section, we present a challenging drone flight task in strong winds, where the drone must follow agile trajectories through narrow gates, which are only slightly wider than the drone. The overall result is depicted in Fig. 3.1 and Video 1. As shown in Fig. 3.1(A), the gates used in our experiments are 110 cm in width, which is only slightly wider than the drone (85 cm wide, 75 cm long). To visualize the trajectory using long-exposure photography, our drone is deployed with four main light emitting diodes (LEDs) on its legs, where the two rear LEDs are red and the front two are white. There are also several small LEDs on the flight controller, the computer, and the motor controllers, which can be seen in the long-exposure shots.

Task Design We tested our method on three different tasks. In the first task (see Fig. 3.1(B,D,F-I) and Video 1), the desired trajectory is a 3 m by 1.5 m figure-8 in the $x-z$ plane with a lap time of 5 s. A gate is placed at the left bottom part of the trajectory. The minimum clearance is about 10 cm, as seen in Fig. 3.1(I), which requires that the controller precisely tracks the trajectory. The maximum speed and acceleration of the desired trajectory are 2.7 m/s and $5.0\,\mathrm{m/s^2}$, respectively. The wind speed is 3.1 m/s. The second task (see Video 1) is the same as the first one, except that it uses a more challenging, time-varying wind condition, $3.1 + 1.8\sin(\frac{2\pi}{5}t)\mathrm{m/s}$. In the third task (see Fig. 3.1(C,E) and Video 1), the desired trajectory is a 3 m by 2.5 m ellipse in the $x-y$ plane with a lap time of 5 s. We placed two gates on the left and right sides of the ellipse. As with the first task, the wind speed is 3.1 m/s.

Performance For all three tasks, we used our primary method, Neural-Fly, where the representation is learned using the dataset collected by the custom-built drone. Figure 3.1(D,E) are two long-exposure photos with an exposure time of 5 s, which is the same as the lap time of the desired trajectory. We see that our method precisely tracked the desired trajectories and flew safely through the gates (see Video 1). These long-exposure photos also captured the smoke visualization of the wind condition. We would like to emphasize that the drone is wider than the LED light region, since the LEDs are located on the legs (see Fig. 3.1(A)). Figure 3.1(F-I) are four high-speed photos with a shutter speed of 1/200s. These four photos captured the moment the drone passed through the gate in the first task, as well as the complex interaction between the drone and the wind. We see that the aerodynamic effects are complex and non-stationary and depend on the UAV attitude, the relative velocity, and aerodynamic interactions between the propellers and the wind.

Outdoor Experiments

We tested our algorithm outdoors in gentle breeze conditions (wind speeds measured up to 17 km/h). An onboard GPS receiver provided position information to the EKF, giving lower precision state estimation, and therefore less precise aerodynamic residual force estimation. Following the same aforementioned figure-8 trajectory, the controller reached 7.5 cm mean tracking error, shown in Fig. 3.7.

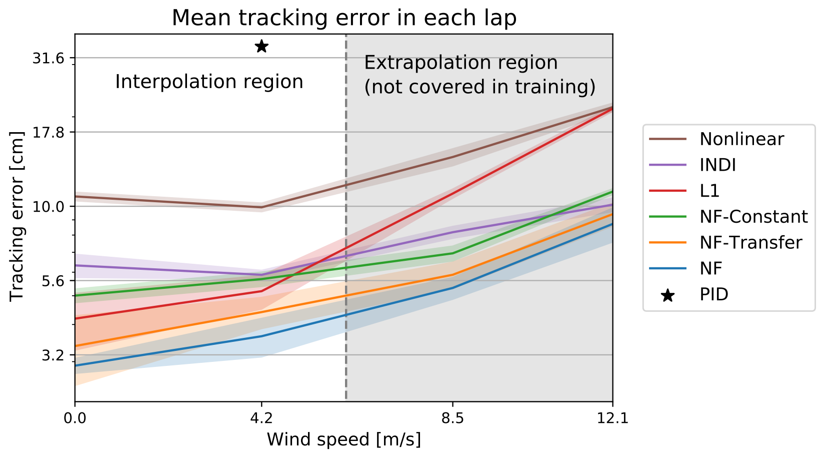


Figure 3.6: **Mean tracking errors of each lap in different wind conditions.** This figure shows position tracking errors of different methods as wind speed increases. Solid lines show the mean error over 6 laps and the shade areas show standard deviation of the mean error on each lap. The gray area indicates the extrapolation region, where the wind speeds are not covered in training. Our primary method (Neural-Fly) achieves state-of-the-art performance even with a strong wind disturbance.

Table 3.1: **Tracking error statistics in** cm **for different wind conditions.** Two metrics are considered: root-mean-square (RMS) and mean.

Method	Wind speed									
	0 m/s		4.2 m/s		8.5 m/s		12.1 m/s		8.5+ 2.4sin(t) m/s	
	RMS	Mean	RMS	Mean	RMS	Mean	RMS	Mean	RMS	Mean
Nonlinear	11.9	10.8	10.7	9.9	16.3	14.7	23.9	21.6	31.2	28.2
INDI	7.3	6.3	6.4	5.9	8.5	8.2	10.7	10.1	11.1	10.3
L1	4.6	4.2	5.8	5.2	12.1	11.1	22.7	21.3	13.0	11.6
NF-Constant	5.4	5.0	6.1	5.7	7.5	6.9	12.7	11.2	12.7	12.1
NF-Transfer	3.7	3.4	4.8	4.4	6.2	5.9	10.2	9.4	8.8	8.0
NF	**3.2**	**2.9**	**4.0**	**3.7**	**5.8**	**5.3**	**9.4**	**8.7**	**7.6**	**6.9**

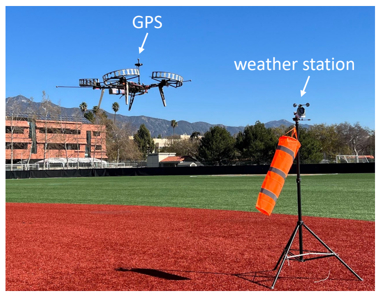
3.3 Discussion

State-of-the-art Tracking Performance

When measuring position tracking errors, we observe that our Neural-Fly method outperforms state-of-the-art flight controllers in all wind conditions. Neural-Fly uses deep-learning to obtain a compact representation of the aerodynamic disturbances and incorporates that representation into an adaptive control design to achieve high precision tracking performance. The benchmark methods used in this article are nonlinear control, INDI, and $\mathcal{L}_1$ and performance is compared tracking an agile figure-8 in constant and time-varying wind speeds up to 43.6 km/h (12.1 m/s). Furthermore, we observe a mean tracking error of 2.9 cm in 0 km/h wind, which is comparable with state-of-the-art tracking performance demonstrated on more aggressive racing drones [4], [7] despite several architectural limitations such as limited control rate in offboard mode, a larger, less maneuverable vehicle, and without direct motor speed measurements. All our experiments were conducted using the standard PX4 attitude controller, with Neural-Fly implemented in an onboard, low cost, and "credit-card sized" Raspberry Pi 4 computer. Furthermore, Neural-Fly is robust to changes in vehicle configuration, as demonstrated by the similar performance of Neural-Fly-Transfer.

To understand the fundamental tracking-error limit, we estimate that the localization precision from the OptiTrack system is about 1 cm, which is a practical lower bound for the average tracking error in our system (see more details in the supplementary material, 4.8). This is based on the fact that the difference between the OptiTrack position measurement and the onboard EKF position estimate is around 1 cm.

To achieve a tracking error of 1 cm, remaining improvements should focus on reducing code execution time, communication delays, and attitude tracking delay. We measured the combined code execution time and communication delay to be at least 15 ms and often as much as 30 ms. A faster implementation (such as using C++ instead of Python) and streamlined communication layer (such as using ROS2's real-time features) could allow us to achieve tracking errors on the order of the localization accuracy. Attitude tracking delay can be substantially reduced through the use of a nonlinear attitude controller (e.g., [45]). Our method is also directly extensible to attitude control because attitude dynamics match the Euler-Lagrange dynamics used in our derivations. However, further work is needed to understand the interaction of the learned dynamics with the cascaded control design when implementing a tracking attitude controller.



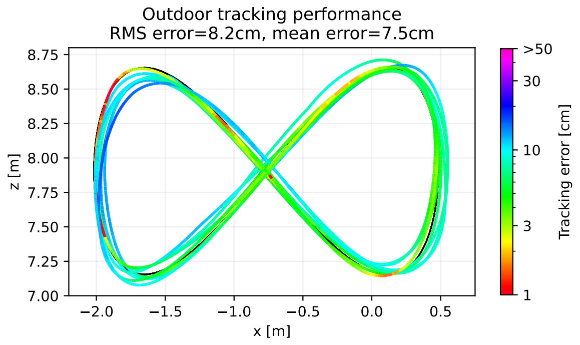


Figure 3.7: **Outdoor flight setup and performance. Left:** In outdoor experiments, a GPS module is deployed for state estimation, and a weather station records wind profiles. The maximum wind speed during the test was around 17 km/h (4.9 m/s). **Right:** Trajectory tracking performance of Neural-Fly.

We have tested our control method in outdoor flight to demonstrate that it is robust to less precise state estimation and does not rely on any particular features of our test facility. Although control and estimation are usually separately designed parts of an autonomous system, aggressive adaptive control requires minimal noise in force measurement to effectively and quickly compensate for unmodeled dynamics. Testing our method in outdoor flight, the quadrotor maintains precise tracking with only 7.5 cm tracking error on a gentle breezy day with wind speeds around 17 km/h.

Challenges Caused by Unknown and Time-varying Wind Conditions

In the real world, the wind conditions are not only unknown but also constantly changing, and the vehicle must continuously adapt. We designed the sinusoidal wind test to emulate unsteady or gusty wind conditions. Although our learned model is trained on static and approximately uniform wind condition data, Neural-Fly can quickly identify changing wind speed and maintains precise tracking even on the sinusoidal wind experiment. Moreover, in each of our experiments, the wind conditions were unknown to the UAV before starting the test yet were quickly identified by the adaptation algorithm.

Our work demonstrated that it is possible to repeatably and quantitatively test quadrotor flight in time-varying wind. Our method separately learns the wind effect's dependence on the vehicle state (i.e., the wind-invariant representation in Fig. 3.2(A)) and the wind condition (i.e., the wind-specific linear weight in Fig. 3.2(A)). This separation allows Neural-Fly to quickly adapt to the time-varying wind even as the UAV follows a dynamic trajectory, with an average tracking error below 8.7 cm in

Table 3.1.

Computational Efficiency of Our Method

In the offline meta-learning phase, the proposed DAIML algorithm is able to learn an effective representation of the aerodynamic effect in a data efficient manner. This requires only 12 minutes of flight data at 50 Hz, for a total of 36,000 data points. The training procedure only takes 5 minutes on a normal desktop computer. In the online adaptation phase, our adaptive control method only takes 10 ms to compute on a compact onboard Linux computer (Raspberry Pi 4). In particular, the feedforward inference time via the learned basis function is about 3.5 ms and the adaptation update is about 3.0 ms, which implies the compactness of the learned representation.

Generalization to New Trajectories and New Aircraft

It is worth noting that our control method is orthogonal to the design of the desired trajectory. In this article, we focus on the figure-8 trajectory which is a commonly used control benchmark. We also demonstrate our method flying a horizontal ellipse during the narrow gate demonstration Fig. 3.1. Note that our method supports any trajectory planners such as [1] or learning-based planners [46]. In particular, for those planners which require a precise and agile downstream controller (e.g., for close-proximity flight or drone racing [1], [10]), our method immediately provides a solution and further pushes the boundary of these planners, because our state-of-the-art tracking capabilities enable tighter configurations and smaller clearances. However, further research is required to understand the coupling between planning and learning-based control near actuation limits. Future work will consider using Neural-Fly in a combined planning and control structure such as MPC, which will be able to handle actuation limits.

The comparison between Neural-Fly and Neural-Fly-Transfer show that our approach is robust to changing vehicle design and the learned representation does not depend on the vehicle. This demonstrates the generalizability of the proposed method running on different quadrotors. Moreover, our control algorithm is formulated generally for all robotic systems described by the Euler-Langrange equation (see “Materials and Methods”), including many types of aircraft such as [21].

3.4 Materials and Methods

Overview

We consider a general robot dynamics model:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = u + f(q, \dot{q}, w) \tag{3.1}$$

where $q, \dot{q}, \ddot{q} \in \mathbb{R}^n$ are the n dimensional position, velocity, and acceleration vectors, $M(q)$ is the symmetric, positive definite inertia matrix, $C(q, \dot{q})$ is the Coriolis matrix, $g(q)$ is the gravitational force vector and $u \in \mathbb{R}^n$ is the control force. Most importantly, $f(q, \dot{q}, w)$ incorporates unmodeled dynamics, and $w \in \mathbb{R}^m$ is an unknown, hidden state used to represent the underlying environmental conditions, which is potentially time-variant. Specifically, in this article, w represents the wind profile (for example, the wind profile in Fig. 3.1), and different wind profiles yield different unmodeled aerodynamic disturbances for the UAV.

Neural-Fly can be broken into two main stages, the offline meta-learning stage and the online adaptive control stage. These two stages build a model of the unknown dynamics of the form

$$f(q, \dot{q}, w) \approx \phi(q, \dot{q})a(w), \tag{3.2}$$

where ϕ is a basis or representation function shared by all wind conditions and captures the dependence of the unmodeled dynamics on the robot state, and a is a set of linear coefficients that is updated for each condition. In the supplementary material (4.2), we prove that the decomposition $\phi(q, \dot{q})a(w)$ exists for any analytic function $f(q, \dot{q}, w)$. In the offline meta-learning stage, we learn ϕ as a DNN using our meta-learning algorithm DAIML. This stage results in learning ϕ as a wind-invariant representation of the unmodeled dynamics, which generalizes to new trajectories and new wind conditions. In the online adaptive control stage, we adapt the linear coefficients a using adaptive control. Our adaptive control algorithm is a type of composite adaptation and was carefully designed to allow for fast adaptation while maintaining the global exponential stability and robustness of the closed loop system. The offline learning and online control architectures are illustrated in Fig. 3.2(B) and Fig. 3.2(A,C), respectively.

Data Collection

To generate training data to learn a wind-invariant representation of the unmodeled dynamics, the drone tracks a randomized trajectory with the baseline nonlinear controller for 2 minutes each in several different static wind conditions. Figure 3.3(A) illustrates one trajectory under the wind condition 13.3 km/h (3.7 m/s). The set of input-output pairs for the k^{th} such trajectory is referred as the k^{th} subdataset, D_{w_k}, with the wind condition w_k. Our dataset consists of 6 different subdatasets with wind speeds from 0 km/h to 22.0 km/h (6.1 m/s), which are in the white interpolation region in Fig. 3.6.

The trajectory follows a polynomial spline between 3 waypoints: the current position and two randomly generated target positions. The spline is constrained to have zero velocity, acceleration, and jerk at the starting and ending waypoints. Once the end of one spline is reached, a new random spline is generated and the process repeats for the duration of the training data flight. This process allows us to generate a large amount of data using a trajectory very different from the trajectories used to test our method, such as the figure-8 in Fig. 3.1. By training and testing on different trajectories, we demonstrate that the learned model generalizes well to new trajectories.

Along each trajectory, we collect time-stamped data $[q, \dot{q}, u]$. Next, we compute the acceleration $\ddot{q}$ by fifth-order numerical differentiation. Combining this acceleration with (3.1), we get a noisy measurement of the unmodeled dynamics, $y = f(x, w) + \epsilon$, where ϵ includes all sources of noise (e.g., sensor noise and noise from numerical differentiation) and $x = [q; \dot{q}] \in \mathbb{R}^{2n}$ is the state. Finally, this allows us to define the dataset, $\mathcal{D} = \{D_{w_1}, \cdots, D_{w_K}\}$, where

$$D_{w_k} = \left\{ x_k^{(i)}, y_k^{(i)} = f(x_k^{(i)}, w_k) + \epsilon_k^{(i)} \right\}_{i=1}^{N_k} \tag{3.3}$$

is the collection of N_k noisy input-output pairs with wind condition w_k. As we discuss in the "Results" section, in order to show DAIML learns a model which can be transferred between drones, we applied this data collection process on both the custom-built drone and the Intel Aero RTF drone.

The Domain Adversarially Invariant Meta-Learning (DAIML) Algorithm

In this section, we will present the methodology and details of learning the representation function ϕ. In particular, we will first introduce the goal of meta-learning, motivate the proposed algorithm DAIML by the observed domain shift problem from the collected dataset, and finally discuss key algorithmic details.

Meta-Learning Goal Given the dataset, the goal of meta-learning is to learn a representation $\phi(x)$, such that for any wind condition w, there exists a latent variable

$a(w)$ which allows $\phi(x)a(w)$ to approximate $f(x,w)$ well. Formally, an optimal representation, ϕ, solves the following optimization problem:

$$\min_{\phi,a_1,\cdots,a_K} \sum_{k=1}^{K} \sum_{i=1}^{N_k} \left\| y_k^{(i)} - \phi(x_k^{(i)}) a_k \right\|^2, \tag{3.4}$$

where $\phi(\cdot) : \mathbf{R}^{2n} \rightarrow \mathbf{R}^{n\times h}$ is the representation function and $a_k \in \mathbf{R}^h$ is the latent linear coefficient. Note that the optimal weight a_k is specific to each wind condition, but the optimal representation ϕ is shared by all wind conditions. In this article, we use a deep neural network (DNN) to represent ϕ. In the supplementary material (Section S2), we prove that for any analytic function $f(x,w)$, the structure $\phi(x)a(w)$ can approximate $f(x,w)$ with an arbitrary precision, as long as the DNN ϕ has enough neurons. This result implies that the ϕ solved from the optimization in (3.4) is a reasonable representation of the unknown dynamics $f(x,w)$.

Domain Shift Problems One challenge of the optimization in (3.4) is the *inherent domain shift* in x caused by the shift in w. Recall that during data collection we have a program flying the drone in different winds. The actual flight trajectories differ vastly from wind to wind because of the wind effect. Formally, the distribution of $x_k^{(i)}$ varies between k because the underlying environment or context w has changed. For example, as depicted by Fig. 3.3(C), the drone pitches into the wind, and the average degree of pitch depends on the wind condition. Note that pitch is only one component of the state x. The domain shift in the whole state x is even more drastic.

Such inherent shifts in x bring challenges for deep-learning. The DNN ϕ may memorize the distributions of x in different wind conditions, such that the variation in the dynamics $\{f(x,w_1), f(x,w_2), \cdots, f(x,w_K)\}$ is reflected via the distribution of x, rather than the wind condition $\{w_1, w_2, \cdots, w_K\}$. In other words, the optimization in (3.4) may lead to *over-fitting* and may not properly find a wind-invariant representation ϕ.

To solve the domain shift problem, inspired by [47], we propose the following adversarial optimization framework:

$$\max_{h} \min_{\phi,a_1,\cdots,a_K} \sum_{k=1}^{K} \sum_{i=1}^{N_k} \left(\left\| y_k^{(i)} - \phi(x_k^{(i)}) a_k \right\|^2 - \alpha \cdot \mathrm{loss}\left(h(\phi(x_k^{(i)})), k \right) \right), \tag{3.5}$$

where h is another DNN that works as a discriminator to predict the environment index out of K wind conditions, $\mathrm{loss}(\cdot)$ is a classification loss function (e.g., the cross entropy loss), $\alpha \geq 0$ is a hyperparameter to control the degree of regularization, k

Algorithm 1: Domain Adversarially Invariant Meta-Learning (DAIML)

```
Hyperparameter: $\alpha \geq 0, 0 < \eta \leq 1, \gamma > 0$
Input: $\mathcal{D} = \{D_{w_1}, \cdots, D_{w_K}\}$
Initialize: Neural networks $\phi$ and $h$
Result: Trained neural networks $\phi$ and $h$
1  repeat
2      Randomly sample $D_{w_k}$ from $\mathcal{D}$
3      Randomly sample two disjoint batches $B^a$ (adaptation set) and $B$ (training
         set) from $D_{w_k}$
4      Solve the least squares problem $a^*(\phi) = \arg\min_a \sum_{i \in B^a} \left\| y_k^{(i)} - \phi(x_k^{(i)}) a \right\|^2$
5      if $\|a^*\| > \gamma$ then
6          $a^* \leftarrow \gamma \cdot \frac{a^*}{\|a^*\|}$    ▷ normalization
7      Train DNN $\phi$ using stochastic gradient descent (SGD) and spectral
         normalization with loss
             $$\sum_{i \in B} \left( \left\| y_k^{(i)} - \phi(x_k^{(i)}) a^* \right\|^2 - \alpha \cdot \mathrm{loss}\left( h(\phi(x_k^{(i)})), k \right) \right)$$
8      if rand() $\leq \eta$ then
9          Train DNN $h$ using SGD with loss $\sum_{i \in B} \mathrm{loss}\left( h(\phi(x_k^{(i)})), k \right)$
10 until convergence
```

is the wind condition index, and (i) is the input-output pair index. Intuitively, h and ϕ play a zero-sum max-min game: the goal of h is to predict the index k directly from $\phi(x)$ (achieved by the outer max); the goal of ϕ is to approximate the label $y_k^{(i)}$ while making the job of h harder (achieved by the inner min). In other words, h is a learned regularizer to remove the environment information contained in ϕ. In our experiments, the output of h is a K−dimensional vector for the classification probabilities of K conditions, and we use the cross entropy loss for loss$(\cdot)$, which is given as

$$\mathrm{loss}\left(h(\phi(x_k^{(i)})), k \right) = -\sum_{j=1}^{K} \delta_{kj} \log \left(h(\phi(x_k^{(i)}))^\top e_j \right) \tag{3.6}$$

where $\delta_{kj} = 1$ if $k = j$ and $\delta_{kj} = 0$ otherwise and e_j is the standard basis function.

Design of the DAIML Algorithm Finally, we solve the optimization problem in (3.5) by the proposed algorithm DAIML (described in Algorithm 1 and illustrated in Fig. 3.2(B)), which belongs to the category of gradient-based meta-learning [31], but with least squares as the adaptation step. DAIML contains three steps: (i) The adaptation step (Line 4-6) solves a least squares problem as a function of ϕ on the adaptation set B^a. (ii) The training step (Line 7) updates the learned representation ϕ on the training set B, based on the optimal linear coefficient a^* solved from the adaptation step. (iii) The regularization step (Line 8-9) updates the discriminator h on the training set.

We emphasize important features of DAIML: (i) After the adaptation step, a^* is a function of ϕ. In other words, in the training step (Line 7), the gradient with respect to the parameters in the neural network ϕ will backpropagate through a^*. Note that the least-square problem (Line 4) can be solved efficiently with a closed-form solution. (ii) The normalization (Line 6) is to make sure $\|a^*\| \leq \gamma$, which improves the robustness of our adaptive control design. We also use spectral normalization in training ϕ, to control the Lipschitz property of the neural network and improve generalizability [8], [10], [12]. (iii) We train h and ϕ in an alternating manner. In each iteration, we first update ϕ (Line 7) while fixing h and then update h (Line 9) while fixing ϕ. However, the probability to update the discriminator h in each iteration is $\eta \leq 1$ instead of 1, to improve the convergence of the algorithm [48].

We further motivate the algorithm design using Fig. 3.3 and Fig. 3.4. Figure 3.3(A,B) shows the input and label from one wind condition, and Fig. 3.3(C) shows the distributions of the pitch component in input and the $x-$component in label, in different wind conditions. The distribution shift in label implies the importance of meta-learning and adaptive control, because the aerodynamic effect changes drastically as the wind condition switches. On the other hand, the distribution shift in input motivates the need of DAIML. Figure 3.4 depicts the evolution of the optimal linear coefficient (a^*) solved from the adaptation step in DAIML, via the t-distributed stochastic neighbor embedding (t-SNE) dimension reduction, which projects the 12-dimensional vector a^* into 2-d. The distribution of a^* is more and more clustered as the number of training epochs increases. In addition, the clustering behavior in Fig. 3.4 has a concrete physical meaning: right top part of the t-SNE plot corresponds to a higher wind speed. These properties imply the learned representation ϕ is indeed shared by all wind conditions, and the linear weight a contains the wind-specific information. Finally, note that ϕ with 0 training epoch reflects random features, which cannot decouple different wind conditions as cleanly as the trained representation ϕ. Similarly, as shown in Fig. 3.4, if we ignore the adversarial regularization term (by setting $\alpha = 0$), different a^* vectors in different conditions are less disentangled, which indicates that the learned representation might be less robust and explainable. For more discussions about α we refer to the supplementary materials (4.3).

Robust Adaptive Controller Design

During the offline meta-training process, a least-squares fit is used to find a set of parameters a that minimizes the force prediction error for each data batch. However, during the online control phase, we are ultimately interested in minimizing the position tracking error, and we can improve the adaptation using a more sophisticated update law. Thus, in this section, we propose a more sophisticated adaptation law for the linear coefficients based upon a Kalman-filter estimator. This formulation results in automatic gain tuning for the update law, which allows the controller to quickly estimate parameters with large uncertainty. We further boost this estimator into a composite adaptation law, that is the parameter update depends both on the prediction error in the dynamics model and on the tracking error, as illustrated in Fig. 3.2. This allows the system to quickly identify and adapt to new wind conditions without requiring persistent excitation. In turn, this enables online adaptation of the high dimensional learned models from DAIML.

Our online adaptive control algorithm can be summarized by the following control law, adaptation law, and covariance update equations, respectively.

$$u_{\mathrm{NF}} = \underbrace{M(q)\ddot{q}_r + C(q,\dot{q})\dot{q}_r + g(q)}_{\text{nominal model feedforward terms}} \underbrace{-Ks}_{\text{PD feedback}} \underbrace{-\phi(q,\dot{q})\hat{a}}_{\text{learning-based feedforward}} \tag{3.7}$$

$$\dot{\hat{a}} = \underbrace{-\lambda\hat{a}}_{\text{regularization term}} \underbrace{-P\phi^\top R^{-1}(\phi\hat{a} - y)}_{\text{prediction error term}} \underbrace{+P\phi^\top s}_{\text{tracking error term}} \tag{3.8}$$

$$\dot{P} = -2\lambda P + Q - P\phi^\top R^{-1}\phi P \tag{3.9}$$

where u_{NF} is the control law, $\dot{\hat{a}}$ is the online linear-parameter update, P is a covariance-like matrix used for automatic gain tuning, $s = \dot{\tilde{q}} + \Lambda\tilde{q}$ is the composite tracking error, y is the measured aerodynamic residual force with measurement noise ϵ, and K, Λ, R, Q, and λ are gains. The structure of this control law is illustrated in Fig. 3.2. Figure 3.2 also shows further quadrotor specific details for the implementation of our method, including the kinematics block, where the desired thrust and attitude are determined from the desired force from (3.7). These blocks are discussed further in the “Implementation Details” section.

In the next section, we will first introduce the baseline control laws, $\bar{u}$ and u_{NL}. Then we discuss our control law u_{NF} in detail. Note that u_{NF} not only depends on the desired trajectory, but also requires the learned representation ϕ and the linear parameter $\hat{a}$ (an estimation of a). The composite adaptation algorithm for $\hat{a}$ is discussed in the following section.

In terms of theoretical guarantees, the control law and adaptation law have been designed so that the closed-loop behavior of the system is robust to imperfect learning and time-varying wind conditions. Specifically, we define $d(t)$ as the representation error: $f = \phi \cdot a + d(t)$, and our theory shows that the robot tracking error exponentially converges to an error ball whose size is proportional to $\|d(t){+}\epsilon\|$ (i.e., the learning error and measurement noise) and $\|\dot{a}\|$ (i.e., how fast the wind condition changes). Later in this section we formalize these claims with the main stability theorem and present a complete proof in the supplementary materials.

Nonlinear Control Law We start by defining some notation. The composite velocity tracking error term s and the reference velocity $\dot{q}_r$ are defined such that

$$s = \dot{q} - \dot{q}_r = \dot{\tilde{q}} + \Lambda\tilde{q} \tag{3.10}$$

where $\tilde{q} = q - q_d$ is the position tracking error and Λ is a positive definite gain matrix. Note when s exponentially converges to an error ball around 0, q will exponentially converge to a proportionate error ball around the desired trajectory $q_d(t)$ (see 4.5). Formulating our control law in terms of the composite velocity error s simplifies the analysis and gain tuning without loss of rigor.

The baseline nonlinear (NL) control law using PID feedback is defined as

$$u_{\mathrm{NL}} = \underbrace{M(q)\ddot{q}_r + C(q,\dot{q})\dot{q}_r + g(q)}_{\text{nonlinear feedforward terms}} \underbrace{-Ks - K_I \int s dt}_{\text{PID feedback}}. \tag{3.11}$$

where K and K_I are positive definite control gain matrices. Note a standard PID controller typically only includes the PI feedback on position error, D feedback on velocity, and gravity compensation. This only leads to local exponential stability about a fixed point, but it is often sufficient for gentle tasks such as a UAV hovering and slow trajectories in static wind conditions. In contrast, this nonlinear controller includes feedback on velocity error and feedforward terms to account for known dynamics and desired acceleration, which allows good tracking of dynamic trajectories in the presence of nonlinearities (e.g., $M(q)$ and $C(q,\dot{q})$ are nonconstant in attitude control). However, this control law only compensates for changing wind conditions and unmodeled dynamics through an integral term, which is slow to react to changes in the unmodeled dynamics and disturbance forces.

Our method improves the controller by predicting the unmodeled dynamics and disturbance forces, and, indeed, in Table 3.1 we see a substantial improvement gained by using our learning method. Given the learned representation of the residual dynamics, $\phi(q, \dot{q})$, and the parameter estimate $\hat{a}$, we replace the integral term with the learned force term, $\hat{f} = \phi\hat{a}$, resulting in our control law in (3.7). Neural-Fly uses ϕ trained using DAIML on a dataset collected with the same drone. Neural-Fly-Transfer uses ϕ trained using DAIML on a dataset collected with a different drone, the Intel Aero RTF drone. Neural-Fly-Constant does not use any learning but instead uses $\phi = I$ and is included to demonstrate that the main advantage of our method comes from the incorporation of learning. The learning-based methods, Neural-Fly and Neural-Fly-Transfer, outperform Neural-Fly-Constant because the compact learned representation can effectively and quickly predict the aerodynamic disturbances online in Fig. 3.5. This comparison is further discussed in the supplementary materials (4.7).

Composite Adaptation Law We define an adaptation law that combines a tracking error update term, a prediction error update term, and a regularization term in (3.8) and (3.9), where y is a noisy measurement of f, λ is a damping gain, P is a covariance matrix which evolves according to (3.9), and Q and R are two positive definite gain matrices. Some readers may note that the regularization term, prediction error term, and covariance update, when taken alone, are in the form of a Kalman-Bucy filter. This Kalman-Bucy filter can be derived as the optimal estimator that minimizes the variance of the parameter error [49]. The Kalman-Bucy filter perspective provides intuition for tuning the adaptive controller: the damping gain λ corresponds to how quickly the environment returns to the nominal conditions, Q corresponds to how quickly the environment changes, and R corresponds to the combined representation error d and measurement noise for y. More discussion on the gain tuning process is included in 4.6. However, naively combining this parameter estimator with the controller can lead to instabilities in the closed-loop system behavior unless extra care is taken in constraining the learned model and tuning the gains. Thus, we have designed our adaptation law to include a tracking error term, making (3.8) a composite adaptation law, guaranteeing stability of the closed-loop system (see Theorem 3.4.1), and in turn simplifying the gain tuning process. The regularization term allows the stability result to be independent of the persistent excitation of the learned model ϕ, which is particularly relevant when using high-dimensional learned representations. The adaptation gain and covariance matrix, P, acts as automatic gain tuning for the adaptive controller, which allows the controller to quickly adapt to when a new mode in the learned model is excited.

Stability and Robustness Guarantees First we formally define the representation error $d(t)$, as the difference between the unknown dynamics $f(q, \dot{q}, w)$ and the best linear weight vector a given the learned representation $\phi(q, \dot{q})$, namely, $d(t) = f(q, \dot{q}, w) - \phi(q, \dot{q})a(w)$. The measurement noise for the measured residual force is a bounded function $\epsilon(t)$ such that $y(t) = f(t) + \epsilon(t)$. If the environment conditions are changing, we consider the case that $\dot{a} \neq 0$. This leads to the following stability theorem.

Theorem 3.4.1. *If we assume that the desired trajectory has bounded derivatives and the system evolves according to the dynamics in (3.1), the control law (3.7), and the adaptation law (3.8) and (3.9), then the position tracking error exponentially converges to the ball*

$$\lim_{t\to\infty} \|\tilde{q}\| \leq \sup_{t} \left[C_1 \|d(t)\| + C_2 \|\epsilon(t)\| + C_3 \left(\lambda \|a(t)\| + \|\dot{a}(t)\|\right) \right], \tag{3.12}$$

where C_1, C_2, and C_3 are three bounded constants depending on $\phi, R, Q, K, \Lambda, M$, and λ.

Implementation Details

Quadrotor Dynamics Now we introduce the quadrotor dynamics. Consider states given by global position, $p \in \mathbb{R}^3$, velocity $v \in \mathbb{R}^3$, attitude rotation matrix $R \in \mathrm{SO}(3)$, and body angular velocity $\omega \in \mathbb{R}^3$. Then dynamics of a quadrotor are

$$\dot{p} = v, \qquad m\dot{v} = mg + Rf_u + f, \tag{3.13a}$$

$$\dot{R} = RS(\omega), \qquad J\dot{\omega} = J\omega \times \omega + \tau_u, \tag{3.13b}$$

where m is the mass, J is the inertia matrix of the quadrotor, $S(\cdot)$ is the skew-symmetric mapping, g is the gravity vector, $f_u = [0, 0, T]^\top$ and $\tau_u = [\tau_x, \tau_y, \tau_z]^\top$ are the total thrust and body torques from four rotors predicted by the nominal model, and $f = [f_x, f_y, f_z]^\top$ are forces resulting from unmodeled aerodynamic effects due to varying wind conditions.

We cast the position dynamics in (3.13a) into the form of (3.1), by taking $M(q) = mI$, $C(q, \dot{q}) \equiv 0$, and $u = Rf_u$. Note that the quadrotor attitude dynamics (3.13b) is also a special case of (3.1) [15], [50], and thus our method can be extended to attitude control. We implement our method in the position control loop, that is we use our method to compute a desired force u_d. Then the desired force is decomposed into the desired attitude R_d and the desired thrust T_d using kinematics (see Fig. 3.2). Then the desired attitude and thrust are sent to the onboard PX4 flight controller.

Neural Network Architectures and Training Details In practice, we found that in addition to the drone velocity v, the aerodynamic effects also depend on the drone attitude and the rotor rotation speed. To that end, the input state x to the deep neural network ϕ is a 11-d vector, consisting of the drone velocity (3-d), the drone attitude represented as a quaternion (4-d), and the rotor speed commands as a pulse width modulation (PWM) signal (4-d) (see Fig. 3.2 and 3.3). The DNN ϕ has four fully-connected hidden layers, with an architecture $11 \rightarrow 50 \rightarrow 60 \rightarrow 50 \rightarrow 4$ and Rectified Linear Units (ReLU) activation. We found that the three components of the wind-effect force, f_x, f_y, f_z, are highly correlated and sharing common features, so we use ϕ as the basis function for all the component. Therefore, the wind-effect force f is approximated by

$$f \approx \begin{bmatrix} \phi(x) & 0 & 0 \\ 0 & \phi(x) & 0 \\ 0 & 0 & \phi(x) \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}, \tag{3.14}$$

where $a_x, a_y, a_z \in \mathbf{R}^4$ are the linear coefficients for each component of the wind-effect force. We followed Algorithm 1 to train ϕ in PyTorch, which is an open source deep-learning framework. We refer to the supplementary material for hyperparameter details (4.3).

Note that we explicitly include the PWM as an input to the ϕ network. The PWM information is a function of $u = Rf_u$, which makes the controller law (e.g., (3.7)) non-affine in u. We solve this issue by using the PWM from the last time step as an input to ϕ, to compute the desired force u_d at the current time step. Because we train ϕ using spectral normalization (see Algorithm 1), this method is stable and guaranteed to converge to a fixed point, as discussed in [8].

Controller Implementation For experiments, we implemented a discrete form of the Neural-Fly controllers, given in 4.4. For INDI, we implemented the position and acceleration controller from Sections III.A and III.B in [4]. For $\mathcal{L}_1$ adaptive control, we followed the adaptation law first presented in [6] and used in [7] and augment the nonlinear baseline control with $\hat{f} = -u_{\mathcal{L}_1}$.

Acknowledgements

We thank M. Anderson for help configuring the quadrotor platform, and M. Anderson and P. Spieler for help troubleshooting hardware. We also thank N. Badillo and L. Pabon Madrid for help in experiments.

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). This research was also conducted in part with funding from Raytheon Technologies. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. The experiments reported in this article were conducted at Caltech's Center for Autonomous Systems and Technologies (CAST).

References

[1] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, Jul. 21, 2021. [Online]. Available: `https://www.science.org/doi/abs/10.1126/scirobotics.abh1221` (visited on 09/08/2021).

[2] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, Apr. 2018, ISSN: 2377-3766. DOI: `10.1109/LRA.2017.2776353`.

[3] P. Ventura Diaz and S. Yoon, "High-fidelity computational aerodynamics of multi-rotor unmanned aerial vehicles," in *2018 AIAA Aerospace Sciences Meeting*, ser. AIAA SciTech Forum, American Institute of Aeronautics and Astronautics, Jan. 7, 2018. DOI: `10.2514/6.2018-1266`. [Online]. Available: `https://arc.aiaa.org/doi/10.2514/6.2018-1266` (visited on 03/27/2023).

[4] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 1203–1218, May 2021, ISSN: 1558-0865. DOI: `10.1109/TCST.2020.3001117`.

[5] S. Mallikarjunan, B. Nesbitt, E. Kharisov, E. Xargay, N. Hovakimyan, and C. Cao, "L1 adaptive controller for attitude control of multirotors," in *AIAA Guidance, Navigation, and Control Conference*, Minneapolis, Minnesota: American Institute of Aeronautics and Astronautics, Aug. 13, 2012, ISBN: 978-1-60086-938-9. DOI: `10.2514/6.2012-4831`. [Online]. Available: `https://arc.aiaa.org/doi/10.2514/6.2012-4831` (visited on 03/04/2022).

[6] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "L1-adaptive MPPI architecture for robust and agile control of multirotors,"

in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 7661–7666. DOI: 10.1109/IROS45743.2020.9341154.

[7] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear MPC for quadrotors," Sep. 9, 2021. arXiv: 2109.04210 [cs]. [Online]. Available: http://arxiv.org/abs/2109.04210 (visited on 09/16/2021).

[8] G. Shi, X. Shi, M. O'Connell, *et al.*, "Neural lander: Stable drone landing control using learned dynamics," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 9784–9790. DOI: 10.1109/ICRA.2019.8794351,

[9] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, "Neural-swarm: Decentralized close-proximity multirotor control using learned interactions," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 3241–3247. DOI: 10.1109/ICRA40945.2020.9196800.

[10] G. Shi, W. Hönig, X. Shi, Y. Yue, and S.-J. Chung, "Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1063–1079, Apr. 2022, ISSN: 1941-0468. DOI: 10.1109/TRO.2021.3098436.

[11] G. Torrente, E. Kaufmann, P. Fohn, and D. Scaramuzza, "Data-driven MPC for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3769–3776, Apr. 2021, ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2021.3061307. [Online]. Available: https://ieeexplore.ieee.org/document/9361343/ (visited on 10/08/2021).

[12] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky, "Spectrally-normalized margin bounds for neural networks," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/b22b257ad0519d4500539da3c8bcf4dd-Abstract.html (visited on 11/18/2022).

[13] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "PIXHAWK -A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1-2, pp. 21–39, Aug. 2012, ISSN: 0929-5593. DOI: 10.1007/s10514-012-9281-4. [Online]. Available: https://graz.pure.elsevier.com/en/publications/pixhawk-a-micro-aerial-vehicle-design-for-autonomous-flight-using (visited on 08/31/2021).

[14] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[15] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, N.J: Prentice Hall, 1991, 459 pp., ISBN: 978-0-13-040890-7.

[16] P. A. Ioannou and J. Sun, *Robust Adaptive Control*. PTR Prentice-Hall, 1996, 856 pp., ISBN: 978-0-13-439100-7. Google Books: `TIYqAQAAMAAJ`.

[17] M. Krstic, I. Κανελλακόπουλος, P. V. Kokotovic, and P. V. Kokotović, *Nonlinear and Adaptive Control Design*. Wiley, Jun. 14, 1995, 592 pp., ISBN: 978-0-471-12732-1. Google Books: `wxkoAQAAMAAJ`.

[18] K. S. Narendra and A. M. Annaswamy, *Stable Adaptive Systems*. Courier Corporation, Jul. 12, 2012, 514 pp., ISBN: 978-0-486-14142-8. Google Books: `CRJhmsAHCUcC`.

[19] J. A. Farrell and M. M. Polycarpou, *Adaptive Approximation Based Control*. John Wiley & Sons, Ltd, 2006, ISBN: 978-0-471-78181-3. DOI: `10.1002/0471781819.fmatter`. [Online]. Available: `https://onlinelibrary.wiley.com/doi/abs/10.1002/0471781819.fmatter` (visited on 08/31/2021).

[20] K. Wise, E. Lavretsky, and N. Hovakimyan, "Adaptive control of flight: Theory, applications, and open problems," in *2006 American Control Conference*, Jun. 2006, 6 pp.-. DOI: `10.1109/ACC.2006.1657677`.

[21] X. Shi, P. Spieler, E. Tang, E.-S. Lupu, P. Tokumaru, and S.-J. Chung, "Adaptive nonlinear control of fixed-wing VTOL with airflow vector sensing," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 5321–5327. DOI: `10.1109/ICRA40945.2020.9197344`.

[22] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in Neural Information Processing Systems*, vol. 20, Curran Associates, Inc., 2007. [Online]. Available: `https://proceedings.neurips.cc/paper/2007/hash/013a006f03dbc5392effeb8f18fda755-Abstract.html` (visited on 11/18/2022).

[23] S. Lale, K. Azizzadenesheli, B. Hassibi, and A. Anandkumar, "Model learning predictive control in nonlinear dynamical systems," in *2021 60th IEEE Conference on Decision and Control (CDC)*, Dec. 2021, pp. 757–762. DOI: `10.1109/CDC45484.2021.9683670`.

[24] J. Nakanishi, J. Farrell, and S. Schaal, "A locally weighted learning composite adaptive controller with structure adaptation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, Sep. 2002, 882–889 vol.1. DOI: `10.1109/IRDS.2002.1041502`.

[25] F.-C. Chen and H. Khalil, "Adaptive control of a class of nonlinear discrete-time systems using neural networks," *IEEE Transactions on Automatic Control*, vol. 40, no. 5, pp. 791–801, May 1995, ISSN: 1558-2523. DOI: `10.1109/9.384214`.

[26] E. N. Johnson and A. J. Calise, "Limited authority adaptive flight control for reusable launch vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 26, no. 6, pp. 906–913, Nov. 2003, ISSN: 0731-5090, 1533-3884. DOI: `10.2514/2.6934`. [Online]. Available: `https://arc.aiaa.org/doi/10.2514/2.6934` (visited on 10/10/2021).

[27] K. Narendra and S. Mukhopadhyay, "Adaptive control using neural networks and approximate models," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 475–485, May 1997, ISSN: 1941-0093. DOI: 10.1109/72.572089.

[28] M. Bisheban and T. Lee, "Geometric adaptive control with neural networks for a quadrotor in wind fields," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 4, pp. 1533–1548, Jul. 2021, ISSN: 1558-0865. DOI: 10.1109/TCST.2020.3006184.

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 7553 May 2015, ISSN: 1476-4687. DOI: 10.1038/nature14539. [Online]. Available: https://www.nature.com/articles/nature14539 (visited on 11/18/2022).

[30] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, PMLR, Jul. 17, 2017, pp. 1126–1135. [Online]. Available: https://proceedings.mlr.press/v70/finn17a.html (visited on 08/31/2021).

[31] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5149–5169, Sep. 2022, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2021.3079209.

[32] G. Shi, K. Azizzadenesheli, M. O'Connell, S.-J. Chung, and Y. Yue, "Meta-adaptive nonlinear control: Theory and algorithms," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10 013–10 025, Dec. 6, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/52fc2aee802efbad698503d28ebd3a1f-Abstract.html (visited on 03/27/2023),

[33] A. Nagabandi, I. Clavera, S. Liu, *et al.*, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," Feb. 27, 2019. arXiv: 1803.11347 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1803.11347 (visited on 08/11/2021).

[34] X. Song, Y. Yang, K. Choromanski, *et al.*, "Rapidly adaptable legged robots via evolutionary meta-learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 3769–3776. DOI: 10.1109/IROS45743.2020.9341571.

[35] S. Belkhale, R. Li, G. Kahn, R. McAllister, R. Calandra, and S. Levine, "Model-based meta-reinforcement learning for flight with suspended payloads," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1471–1478, Apr. 2021, ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3057046.

[36] C. D. McKinnon and A. P. Schoellig, "Meta learning with paired forward and inverse models for efficient receding horizon control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3240–3247, Apr. 2021, ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3063957.

[37] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," in *Proceedings of The 2nd Conference on Robot Learning*, PMLR, Oct. 23, 2018, pp. 617–629. [Online]. Available: `https://proceedings.mlr.press/v87/clavera18a.html` (visited on 11/18/2022).

[38] M. O'Connell, G. Shi, X. Shi, and S.-J. Chung. "Meta-learning-based robust adaptive flight control under uncertain wind conditions." arXiv: `2103.01932`. (Mar. 4, 2021), [Online]. Available: `http://arxiv.org/abs/2103.01932` (visited on 09/23/2021), preprint.

[39] S. Richards, N. Azizan, J.-J. Slotine, and M. Pavone, "Adaptive-control-oriented meta-learning for nonlinear systems," in *Robotics: Science and Systems XVII*, Robotics: Science and Systems Foundation, Jul. 12, 2021, ISBN: 978-0-9923747-7-8. DOI: `10.15607/RSS.2021.XVII.056`. [Online]. Available: `http://www.roboticsproceedings.org/rss17/p056.pdf` (visited on 03/15/2023).

[40] M. Peng, B. Zhu, and J. Jiao. "Linear representation meta-reinforcement learning for instant adaptation." arXiv: `2101.04750 [cs, stat]`. (Jan. 12, 2021), [Online]. Available: `http://arxiv.org/abs/2101.04750` (visited on 11/18/2022), preprint.

[41] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, eabc5986, Oct. 21, 2020, ISSN: 2470-9476. DOI: `10.1126/scirobotics.abc5986`. arXiv: `2010.11251`. [Online]. Available: `http://arxiv.org/abs/2010.11251` (visited on 08/18/2021).

[42] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 23–30. DOI: `10.1109/IROS.2017.8202133`.

[43] F. Ramos, R. Possas, and D. Fox, "BayesSim: Adaptive domain randomization via probabilistic inference for robotics simulators," in *Robotics: Science and Systems XV*, Robotics: Science and Systems Foundation, Jun. 22, 2019, ISBN: 978-0-9923747-5-4. DOI: `10.15607/RSS.2019.XV.029`. [Online]. Available: `http://www.roboticsproceedings.org/rss15/p29.pdf` (visited on 11/18/2022).

[44] D. Morgan, S.-J. Chung, and F. Y. Hadaegh, "Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and model predictive control," in *AIAA Guidance, Navigation, and Control Conference*, ser. AIAA SciTech Forum, American Institute of Aeronautics and Astronautics, Jan. 2, 2015. DOI: `10.2514/6.2015-0599`. [Online]. Available: `https://arc.aiaa.org/doi/10.2514/6.2015-0599` (visited on 03/06/2022).

[45] X. Shi, K. Kim, S. Rahili, and S.-J. Chung, "Nonlinear control of autonomous flying cars with wings and distributed electric propulsion," in *2018 IEEE Conference on Decision and Control (CDC)*, Miami Beach, FL: IEEE, Dec. 2018, pp. 5326–5333, ISBN: 978-1-5386-1395-5. DOI: 10.1109/CDC.2018.8619578. [Online]. Available: https://ieeexplore.ieee.org/document/8619578/ (visited on 08/25/2020).

[46] Y. K. Nakka, A. Liu, G. Shi, A. Anandkumar, Y. Yue, and S.-J. Chung, "Chance-constrained trajectory optimization for safe exploration and learning of non-linear systems," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 389–396, Apr. 2021, ISSN: 2377-3766. DOI: 10.1109/LRA.2020.3044033.

[47] Y. Ganin, E. Ustinova, H. Ajakan, *et al.* "Domain-adversarial training of neural networks." arXiv: 1505.07818 [cs, stat]. (May 26, 2016), [Online]. Available: http://arxiv.org/abs/1505.07818 (visited on 08/23/2022), preprint.

[48] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html (visited on 11/18/2022).

[49] R. E. Kalman and R. S. Bucy, "New results in linear filtering and prediction theory," *Journal of Basic Engineering*, vol. 83, no. 1, pp. 95–108, Mar. 1, 1961, ISSN: 0021-9223. DOI: 10.1115/1.3658902. [Online]. Available: https://asmedigitalcollection.asme.org/fluidsengineering/article/83/1/95/426820/New-Results-in-Linear-Filtering-and-Prediction (visited on 09/21/2021).

[50] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, 1st ed. CRC Press, 1994, ISBN: 978-1-315-13637-0. DOI: 10.1201/9781315136370. [Online]. Available: https://www.taylorfrancis.com/books/9781351469791 (visited on 09/21/2021).

Chapter 4

ANALYSIS, PROOFS, AND IMPLEMENTATION OF NEURAL-FLY

This chapter provides additional details on the methods presented in Chapter 3, including important theoretical results regarding the expressiveness of the DAIML-learning algorithm in Sec. 4.2 and the stability and robustness of the online adaptation algorithm in Sec. 4.5.

4.1 Drone Configuration Details

Table 4.1 presents the configuration information of the custom-built drone (Fig. 3.1(A)) and the Intel Aero drone. We use both drones for data collection and use the custom-built drone exclusively for experiments.

	Custom-built drone	Intel Aero drone
Weight	2.53 kg	1.47 kg
Thrust-to-weight ratio	2.2	1.6
Rotor tilt angle	$12°$ front, $10°$ rear	$0°$
Diameter	85 cm wide, 75 cm long	52 cm wide, 52 cm long
Configuration	Wide-X4	X4
On-board computer	Raspberry Pi 4	Intel Aero computing board (Atom x7 processor)
Flight controller	Pixhawk 4 running PX4	Aero Flight Controller running PX4

Table 4.1: **Drone configuration details.** Configurations of the custom-built drone and the Intel Aero drone with propeller guards.

Precision tracking for drones often relies on specialized hardware and optimized vehicle design, whereas our method achieves precise tracking using improved dynamics prediction through online learning. Although most researchers report the numeric tracking error of their method, it can be difficult to disentangle the improvement of the controller resulting from the algorithmic advancement versus the improvement from specialized hardware. For example moment of inertia generally scales with the radius squared and the lever arm for the motors scales with the radius, so the attitude maneuverability roughly scales with the inverse of the vehicle radius. Similarly, high thrust to weight ratio provides more attitude control author-

	Neural-Fly	INDI [1]	Differentially flat linear drag [2]	Gaussian Process MPC [3]
Flight computer	Raspberry Pi 4	–	laptop	laptop
Flight controller	Pixhawk 4	STM32H7 (400 MHz)	Raceflight Revolt	?
Flight controller firmware	PX4	custom	?	?
Mass [kg]	2.53	0.609	0.610	0.8
Total width [cm]	85	?	?	?
Propeller diameter [in]	11	5	6	?
Motor Spacing [cm]	39*	18	–	?
Thrust-to-weight ratio [-]	2.2	?	4	5
Motion capture frequency [Hz]	100	360	200	100
MPC control frequency [Hz]	–	–	–	50
Position control frequency [Hz]	50	?	55	?
Attitude control frequency [Hz]	<1000	2000	4000	?
Motor speed feedback	No	Optical encoders (5 kHz)	No	No

? indicates information not provided
– indicates information not applicable
* front to back

Table 4.2: **Hardware comparison.** Hardware configuration comparison with other quadrotors that demonstrate state-of-the-art trajectory tracking. Direct comparisons of performance are difficult due to the varying configurations, controller tuning, and flight arenas. However, most methods require extremely maneuverable quadrotors and onboard/offboard computation power to achieve state-of-the-art performance, while Neural-Fly achieves state-of-the-art performance on more standard hardware with all control running onboard.

ity during high acceleration maneuvers. More powerful motors, electronic speed controllers, and batteries together allow faster motor response time further improving maneuverability. Thus, state-of-the-art (SOTA) tracking performance usually requires specialized hardware often used for racing drones, resulting in a vehicle with greater maneuverability than our platform, a higher thrust to weight ratio, and using high-rate controllers sometimes even including direct motor RPM control. In contrast, our custom drone is more representative of typical consumer drone hardware. A detailed comparison with the hardware from some recent work in agile flight control is provided in Table 4.2.

4.2 The Expressiveness of the Learning Architecture

In this section, we theoretically justify the decomposition $f(x, w) \approx \phi(x)a(w)$. In particularly, we prove that any analytic function $\bar{f}(x, w) : [-1, 1]^n \times [-1, 1]^m \to \mathbf{R}$ can be split into a w-invariant part $\bar{\phi}(x)$ and a w-dependant part $\bar{a}(w)$ in the structure $\bar{\phi}(x)\bar{a}(w)$ with arbitrary precision ϵ, where $\bar{\phi}(x)$ and $\bar{a}(w)$ are two polynomials. Further, the dimension of $\bar{a}(w)$ only scales polylogarithmically with $1/\epsilon$.

We first introduce the following multivariate polynomial approximation lemma in the hypercube proved in [4].

Lemma 4.2.1. *(Multivariate polynomial approximation in the hypercube) Let $\bar{f}(x, w) : [-1, 1]^n \times [-1, 1]^m \to \mathbf{R}$ be a smooth function of $[x, w] \in [-1, 1]^{n+m}$ for $n, m \geq 1$. Assume $\bar{f}(x, w)$ is analytic for all $[x, w] \in \mathbf{C}^{n+m}$ with $\Re(x_1^2 + \cdots + x_n^2 + w_1^2 + \cdots + w_m^2) \geq -t^2$ for some $t > 0$, where $\Re(\cdot)$ denotes the real part of a complex number. Then $\bar{f}$ has a uniformly and absolutely convergent multivariate Chebyshev series*

$$\sum_{k_1=0}^{\infty} \cdots \sum_{k_n=0}^{\infty} \sum_{l_1=0}^{\infty} \cdots \sum_{l_m=0}^{\infty} b_{k_1,\cdots,k_n,l_1,\cdots,l_m} T_{k_1}(x_1) \cdots T_{k_n}(x_n) T_{l_1}(w_1) \cdots T_{l_m}(w_m).$$

Define $s = [k_1, \cdots, k_n, l_1, \cdots, l_m]$. The multivariate Chebyshev coefficients satisfy the following exponential decay property:

$$b_s = O\left((1+t)^{-\|s\|_2}\right).$$

Note that this lemma shows that the truncated Chebyshev expansions

$$C_p = \sum_{k_1=0}^{p} \cdots \sum_{k_n=0}^{p} \sum_{l_1=0}^{p} \cdots \sum_{l_m=0}^{p} b_{k_1,\cdots,k_n,l_1,\cdots,l_m} T_{k_1}(x_1) \cdots T_{k_n}(x_n) T_{l_1}(w_1) \cdots T_{l_m}(w_m)$$

will converge to $\bar{f}$ with the rate $O((1+t)^{-p\sqrt{n+m}})$ for some $t > 0$, i.e., $\sup_{[x,w]\in[-1,1]^{n+m}} \|\bar{f}(x, w) - C_p(x, w)\| \leq O((1+t)^{-p\sqrt{n+m}})$. Finally, we are ready to present the following representation theorem.

Theorem 4.2.2. *$\bar{f}(x, w)$ is a function satisfying the assumptions in Lemma 4.2.1. For any $\epsilon > 0$, there exist $h \in \mathbf{Z}^+$, and two Chebyshev polynomials $\bar{\phi}(x) : [-1, 1]^n \to \mathbf{R}^{1\times h}$ and $\bar{a}(w) : [-1, 1]^m \to \mathbf{R}^{h\times 1}$ such that*

$$\sup_{[x,w]\in[-1,1]^{n+m}} \|\bar{f}(x, w) - \bar{\phi}(x)\bar{a}(w)\| \leq \epsilon$$

and $h = O((\log(1/\epsilon))^m)$.

Proof. First note that there exists $p = O\left(\frac{\log(1/\epsilon)}{\sqrt{n+m}}\right)$ such that $\sup_{[x,w]\in[-1,1]^{n+m}} \left\|\bar{f}(x,w) - C_p(x,w)\right\| \le \epsilon$. To simplify the notation, define

$$\begin{aligned} g(x,k,l) &= g(x_1,\cdots,x_n,k_1,\cdots,k_n,l_1,\cdots,l_m) \\ &= b_{k_1,\cdots,k_n,l_1,\cdots,l_m} T_{k_1}(x_1)\cdots T_{k_n}(x_n) \\ g(w,l) &= g(w_1,\cdots,w_m,l_1,\cdots,l_m) = T_{l_1}(w_1)\cdots T_{l_n}(w_m). \end{aligned}$$

Then we have

$$C_p(x,w) = \sum_{k_1,\cdots,k_n=0}^{p} \sum_{l_1,\cdots,l_m=0}^{p} g(x,k_1,\cdots,k_n,l_1,\cdots,l_m) g(w,l_1,\cdots,l_m).$$

Then we rewrite C_p as $C_p(x,w) = \bar{\phi}(x)\bar{a}(w)$:

$$\bar{\phi}(x)^\top = \begin{bmatrix} \sum_{k_1,\cdots,k_n=0}^{p} g(x,k_1,\cdots,k_n,l=[0,0,\cdots,0]) \\ \sum_{k_1,\cdots,k_n=0}^{p} g(x,k_1,\cdots,k_n,l=[1,0,\cdots,0]) \\ \sum_{k_1,\cdots,k_n=0}^{p} g(x,k_1,\cdots,k_n,l=[2,0,\cdots,0]) \\ \vdots \\ \sum_{k_1,\cdots,k_n=0}^{p} g(x,k_1,\cdots,k_n,l=[p,p,\cdots,p]) \end{bmatrix}$$

$$\bar{a}(w) = \begin{bmatrix} g(w,l=[0,0,\cdots,0]) \\ g(w,l=[1,0,\cdots,0]) \\ g(w,l=[2,0,\cdots,0]) \\ \vdots \\ g(w,l=[p,p,\cdots,p]) \end{bmatrix}.$$

Note that the dimension of $\bar{\phi}(x)$ and $\bar{a}(w)$ is

$$h = (p+1)^m = O\left(\left(1 + \frac{\log(1/\epsilon)}{\sqrt{n+m}}\right)^m\right) = O\left((\log(1/\epsilon))^m\right).$$

□

Note that Theorem 4.2.2 can be generalized to vector-valued functions with bounded input space straightforwardly. Finally, since deep neural networks are universal approximators for polynomials [5], Theorem 4.2.2 immediately guarantees the expressiveness of our learning structure, i.e., $\phi(x)a(w)$ can approximate $f(x,w)$ with arbitrary precision, where $\phi(x)$ is a deep neural network and $\hat{\mathbf{a}}$ includes the linear coefficients for all the elements of f. In experiments, we show that a four-layer neural network can efficiently learn an effective representation for the underlying unknown dynamics $f(x,w)$.

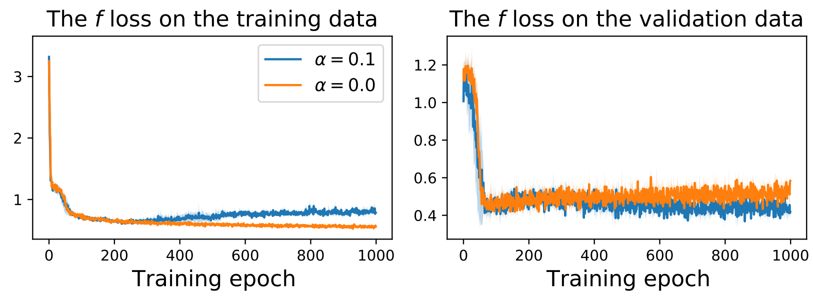


Figure 4.1: **Training and validation loss.** The evolution of the f loss on the training data and validation data in the training process, from three random seeds. Both mean (the solid line) and standard deviation (in the shaded area) are presented. Training with the adversarial regularization term ($\alpha = 0.1$) has similar behaviors as $\alpha = 0$ (no regularization) in the early phase before 300 training epochs, except that it converges slightly faster. However, the regularization term effectively avoids over-fitting and has smaller error on the validation dataset after 300 training epochs.

4.3 Hyperparameters for DAIML and the Interpretation

We implemented DAIML (Algorithm 1) using PyTorch, with hyperparameters reported in Table 4.3. We iteratively tuned these hyperparameters by trial and error. We notice that the behavior of the learning algorithm is not sensitive to most of the parameters in Table 4.3. The training process is shown in Fig. 4.1, where we present the f loss curve on both training set and validation set using three random seeds. The f loss is defined by $\sum_{i\in B} \|y_k^{(i)} - \phi(x_k^{(i)})a^*\|^2$ (see Line 7 in Algorithm 1), which reflects how well ϕ can approximate the unknown dynamics $f(x, w)$. The validation set we considered is from the figure-8 trajectory tracking tasks using the PID and nonlinear baseline methods. Note that the training set consists of a very different set of trajectories (using random waypoint tracking, see Results) , and this difference is for studying whether and when the learned model ϕ starts over-fitting during the training process.

We emphasize a few important parameters as follows. (i) The frequency $0 < \eta \leq 1$ is to control how often the discriminator h is updated. Note that $\eta = 1$ corresponds to the case that ϕ and h are both updated in each iteration. We use $\eta = 0.5$ for training stability, which is also commonly used in training generative adversarial networks [6]. (ii) The regularization parameter $\alpha \geq 0$. Note that $\alpha = 0$ corresponds to the non-adversarial meta-learning case which does not incorporate the adversarial regularization term in (3.5). From Fig. 4.1, clearly a proper choice of α can effectively avoid over-fitting. Moreover, another benefit of having $\alpha > 0$ is that the learned model is more explainable. As observed in Fig. 3.4, $\alpha > 0$ disentangles the linear

Architecture of ϕ net	$11 \to 50 \to 60 \to 50 \to 4$ with ReLU activation functions
Architecture of h net	$4 \to 128 \to 6$ with ReLU activation functions
Batch size of B_a	128
Batch size of B	256
Loss function for h	Cross-entropy loss
Learning rate for training ϕ	0.0005
Learning rate for training h	0.001
Discriminator training frequency, η	0.5
Normalization constant, γ	10
Regularization constant, α	0.1

Table 4.3: **Hyperparameters used in DAIML.**

coefficients a^* between wind conditions. However, if α is too high it may degrade the prediction performance, so we recommend using relatively small value for α such as 0.1.

The importance of having a domain-invariant representation. We use the following example to illustrate the importance of having a domain-invariant representation $\phi(x)$ for online adaptation. Suppose the data distribution in wind conditions 1 and 2 are $P_1(x)$ and $P_2(x)$, respectively, and they do not overlap. Ideally, we would hope these two conditions share an invariant representation and the latent variables are distinct ($a^{(1)}$ and $a^{(2)}$ in the first line in Fig. 4.2 shown below). However, because of the expressiveness of DNNs, ϕ may memorize P_1 and P_2 and learn two modes $\phi_1(x)$ and $\phi_2(x)$. In the second line in the following figure, ϕ_1 and ϕ_2 are triggered if x is in P_1 and P_2, respectively ($\mathbf{1}_{x\in P_1}$ and $\mathbf{1}_{x\in P_2}$ are indicator functions), such that the latent variable a is identical in both wind conditions. Such an overfitted ϕ is not robust and not generalizable: for example, if the drone flies to P_1 in wind condition 2, the wrong mode ϕ_1 will be triggered.

The key idea to tackle this challenge is to encourage diversity in the latent space, which is why we introduced a discriminator in DAIML. Figure 3.4 shows DAIML indeed makes the latent space much more disentangled.

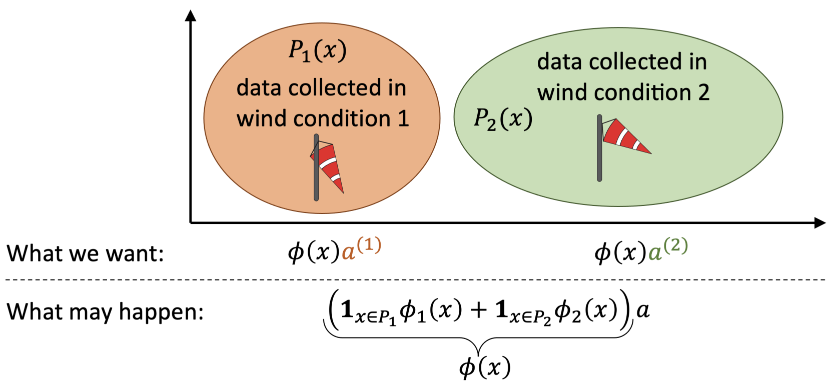

$P_1(x)$
data collected in wind condition 1
data collected in wind condition 2
$P_2(x)$
What we want:
$\phi(x)a^{(1)}$
$\phi(x)a^{(2)}$
What may happen:
$\underbrace{\left(\mathbf{1}_{x\in P_1}\phi_1(x) + \mathbf{1}_{x\in P_2}\phi_2(x)\right)}_{\phi(x)} a$

Figure 4.2: **Importance of domain-invariant representation.**

4.4 Discrete Version of the Proposed Controller

In practice, we implement Neural-Flyon a digital system, and therefore, we require a discrete version of the controller. The feedback control policy u remains the same as presented in the main body of this article. However, the adaptation law must be integrated, and therefore we must be concerned with both the numerical accuracy and computation time of this integration, particularly for the covariance matrix P. During the development of our algorithm, we observed that a naive one-step Euler integration of the continuous time adaptation law would sometimes result P becoming non-positive-definite due to a large $\dot{P}$ magnitude and a coarse integration step size (see [7] for more discussion on the positive definiteness of numerical integration of the differential Riccati equation). To avoid this issue, we instead implemented the adaptation law in two discrete steps, a propagation and an update step, summarized as below. We denote the time at step k as t_k, the value of a parameter before the update step but after the propagation step with a subscript t_k^-, and the value after both the propagation and update step with a subscript t_k^+. The value used in the controller is the value after both the propagation and update steps, that is $\hat{a}(t_k) = \hat{a}_{t_k^+}$. During the propagation step in (4.1) and (4.2) both $\hat{\mathbf{a}}$ and P are regularized. Then, in the update step in (4.4) and (4.5), P and $\hat{\mathbf{a}}$ are updated according to the gain in (4.3). This mirrors a discrete Kalman filter implementation [8] with the tracking error term added in the update step. The discrete Kalman filter exactly integrates the continuous time Kalman filter when the prediction error e, tracking error s, and learned basis functions ϕ are constant between time steps ensuring the positive definiteness of P.

$$\hat{a}_{t_k^-} = \underbrace{(1-\lambda\Delta t_k)}_{\text{damping}}\hat{a}_{t_{k-1}^+} \tag{4.1}$$

$$P_{t_k^-} = (1-\lambda\Delta t_k)^2 P_{t_{k-1}^+} + Q\Delta t_k \tag{4.2}$$

$$K_{t_k} = P_{t_k^-}\phi_{t_k}^\top\left(\phi_{t_k}P_{t_k^-}\phi_{t_k}^\top + R\Delta t_k\right)^{-1} \tag{4.3}$$

$$\hat{a}_{t_k^+} = \hat{a}_{t_k^-} - \underbrace{K_{t_k}\left(\phi_{t_k}\hat{a}_{t_k^-} - y_{t_k}\right)}_{\text{prediction error adaptation}} - \underbrace{P_{t_k^-}\phi_{t_k}^\top s_{t_k}}_{\text{tracking error adaptation}} \tag{4.4}$$

$$P_{t_k^+} = \left(I - K_{t_k}\phi_{t_k}\right)P_{t_k^-}\left(I - K_{t_k}\phi_{t_k}\right)^\top + K_{t_k}R\Delta t_k K_{t_k}^\top \tag{4.5}$$

4.5 Stability and Robustness Formal Guarantees and Proof

We divide the proof of (3.12) into two steps. First, in Theorem 4.5.1, we show that the combined composite velocity tracking error and adaptation error, $\|[s;\tilde{a}]\|$, exponentially converges to a bounded error ball. This implies the exponential convergence of s. Then in Corollary 4.5.1.1 we show that when s is exponentially bounded, $\tilde{q}$ is also exponentially bounded. Combining the exponential bound from Theorem 4.5.1 and the ultimate bound from Corollary 4.5.1.1 proves Theorem 3.4.1.

Before discussing the main proof, let us consider the robustness properties of the feedback controller without considering any specific adaptation law. Taking the dynamics (3.1), control law (3.7), the composite velocity error definition (3.10), and the parameter estimation error $\tilde{a} = \hat{a} - a$, we find

$$M\dot{s} + (C+K)s = -\phi\tilde{a} + d \tag{4.6}$$

We can use the Lyapunov function $\mathcal{V} = s^\top M s$ under the assumption of bounded $\tilde{a}$ to show that

$$\lim_{t\to\infty}\|s\| \le \frac{\sup_t\|d-\phi\tilde{a}\|\lambda_{\max}(M)}{\lambda_{\min}(K)\lambda_{\min}(M)} \tag{4.7}$$

Taking this results alone, one might expect that any online estimator or learning algorithm will lead to good performance. However, the boundedness of $\tilde{a}$ is not guaranteed; Slotine and Li discuss this topic thoroughly [9]. In the full proof below, we show the stability and robustness of the Neural-Fly adaptation algorithm.

First, we introduce the parameter measurement noise $\bar{\epsilon}$, where $\bar{\epsilon} = y - \phi a$. Thus, $\bar{\epsilon} = \epsilon + d$ and $\|\bar{\epsilon}\| \le \|\epsilon\| + \|d\|$ by the triangle inequality. Using the above closed loop dynamics (4.6), the parameter estimation error $\tilde{a}$, and the adaptation law (3.8) and (3.9), the combined velocity and parameter-error closed-loop dynamics are given by

$$\begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} \dot{s} \\ \dot{\tilde{a}} \end{bmatrix} + \begin{bmatrix} C+K & \phi \\ -\phi^T & \phi^\top R^{-1}\phi + \lambda P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} = \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix} \tag{4.8}$$

$$\frac{d}{dt}\left(P^{-1}\right) = -P^{-1}\dot{P}P^{-1} = P^{-1}\left(2\lambda P - Q + P\phi^\top R^{-1}\phi P\right)P^{-1} \tag{4.9}$$

For our stability proof, we rely on the fact that P^{-1} is both uniformly positive definite and uniformly bounded, that is, there exists some positive definite, constant matrices A and B such that $A \succeq P^{-1} \succeq B$. Dieci and Eirola [7] show the slightly weaker result that P is positive definite and finite when ϕ is bounded under the looser assumption $Q \succeq 0$. Following the proof from [7] with the additional assumption that Q is uniformly positive definite, one can show the uniform definiteness and uniform boundedness of P. Hence, P^{-1} is also uniformly positive definite and uniformly bounded.

Theorem 4.5.1. *Given dynamics that evolve according to (4.8) and (4.9), uniform positive definiteness and uniform boundedness of P^{-1}, the norm of $\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}$ exponentially converges to the bound given in (4.10) with rate α.*

$$\lim_{t\to\infty} \left\| \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \right\| \leq \frac{1}{\alpha \lambda_{\min}(\mathcal{M})} \left(\sup_t \|d\| + \sup_t(\|\phi^\top R^{-1}\bar{\epsilon}\|) + \lambda_{\max}(P^{-1}) \sup_t(\|\lambda a + \dot{a}\|) \right) \tag{4.10}$$

where α and $\mathcal{M}$ are functions of ϕ, R, Q, K, M and λ, and $\lambda_{\min}(\cdot)$ and $\lambda_{\max}(\cdot)$ are the minimum and maximum eigenvalues of $(\cdot)$ over time, respectively. Given Corollary 4.5.1.1 and (4.10), the bound in (3.12) is proven. Note $\lambda_{\max}(P^{-1}) = 1/\lambda_{\min}(P)$ and a sufficiently large value of $\lambda_{\min}(P)$ will make the RHS of (4.10) small.

Proof. Now consider the Lyapunov function $\mathcal{V}$ given by

$$\mathcal{V} = \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \tag{4.11}$$

This Lyapunov function has the derivative

$$\dot{\mathcal{V}} = 2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} \dot{s} \\ \dot{\tilde{a}} \end{bmatrix} + \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} \dot{M} & 0 \\ 0 & \frac{d}{dt}\left(P^{-1}\right) \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \tag{4.12}$$

$$= -2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} C+K & \phi \\ -\phi^T & \phi^\top R^{-1}\phi + \lambda P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} + 2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix}$$
$$+ \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} \dot{M} & 0 \\ 0 & \frac{d}{dt}\left(P^{-1}\right) \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \tag{4.13}$$

$$= -2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} K & \phi \\ -\phi^T & \phi^\top R^{-1}\phi + \lambda P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} + 2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix}$$
$$+ \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} 0 & 0 \\ 0 & 2\lambda P^{-1} - P^{-1}QP^{-1} + \phi^\top R^{-1}\phi \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \tag{4.14}$$

$$= -\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} 2K & 0 \\ 0 & \phi^\top R^{-1}\phi + P^{-1}QP^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} + 2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix} \tag{4.15}$$

where we used the fact $\dot{M} - 2C$ is skew-symmetric. As K, $P^{-1}QP^{-1}$, M, and P^{-1} are all uniformly positive definite and uniformly bounded, and $\phi^\top R^{-1}\phi$ is positive semidefinite, there exists some $\alpha > 0$ such that

$$-\begin{bmatrix} 2K & 0 \\ 0 & \phi^\top R^{-1}\phi + P^{-1}QP^{-1} \end{bmatrix} \preceq -2\alpha \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \tag{4.16}$$

for all t.

Define an upper bound for the disturbance term D as

$$D = \sup_t \left\| \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix} \right\| \tag{4.17}$$

and define the function $\mathcal{M}$,

$$\mathcal{M} = \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \tag{4.18}$$

By (4.16), the Cauchy-Schwartz inequality, and the definition of the minimum eigenvalue, we have the following inequality for $\dot{\mathcal{V}}$:

$$\dot{\mathcal{V}} \le -2\alpha\mathcal{V} + 2\sqrt{\frac{\mathcal{V}}{\lambda_{\min}(\mathcal{M})}} D \tag{4.19}$$

Consider the related systems, $\mathcal{W}$ where $\mathcal{W} = \sqrt{\mathcal{V}}$, $2\dot{\mathcal{W}}\mathcal{W} = \dot{\mathcal{V}}$, and the following three equations hold

$$2\dot{\mathcal{W}}\mathcal{W} \leq -2\alpha\mathcal{W}^2 + \frac{2D\mathcal{W}}{\sqrt{\lambda_{\min}(\mathcal{M})}} \tag{4.20}$$

$$\dot{\mathcal{W}} \leq -\alpha\mathcal{W} + \frac{D}{\sqrt{\lambda_{\min}(\mathcal{M})}} \tag{4.21}$$

By the Comparison Lemma [10],

$$\sqrt{\mathcal{V}} = \mathcal{W} \leq \mathrm{e}^{-\alpha t}\left(\mathcal{W}(0) - \frac{D}{\alpha\sqrt{\lambda_{\min}(\mathcal{M})}}\right) + \frac{D}{\alpha\sqrt{\lambda_{\min}(\mathcal{M})}} \tag{4.22}$$

and the stacked state exponentially converges to the ball

$$\lim_{t\to\infty}\left\|\begin{bmatrix} s \\ \tilde{a}\end{bmatrix}\right\| \leq \frac{D}{\alpha\lambda_{\min}(\mathcal{M})} \tag{4.23}$$

This completes the proof. □

Next, we present a corollary which shows the exponential convergence of $\tilde{q}$ when s is exponentially stable.

Corollary 4.5.1.1. *If $\|s(t)\| \leq A\exp(-\alpha t) + B/\alpha$ for some constants A, B, and α, and $s = \dot{\tilde{q}} + \Lambda\tilde{q}$, then*

$$\|\tilde{q}\| \leq \mathrm{e}^{-\lambda_{\min}(\Lambda)t}\|\tilde{q}(0)\| + \int_0^t \mathrm{e}^{-\lambda_{\min}(\Lambda)(t-\tau)}A\mathrm{e}^{-\alpha\tau}\mathrm{d}\tau + \int_0^t \mathrm{e}^{-\lambda_{\min}(\Lambda)(t-\tau)}\frac{B}{\alpha}\mathrm{d}\tau \tag{4.24}$$

thus $\|\tilde{q}\|$ exponentially approaches the bound

$$\lim_{t\to\infty}\|\tilde{q}\| \leq \frac{B}{\alpha\lambda_{\min}(\Lambda)} \tag{4.25}$$

Proof. From the Comparison Lemma [10], we can easily show (4.24). This can be further reduced as follows.

$$\|\tilde{q}\| \leq \mathrm{e}^{-\lambda_{\min}(\Lambda)t}\|\tilde{q}(0)\| + A\mathrm{e}^{-\lambda_{\min}(\Lambda)t}\int_0^t \mathrm{e}^{(\lambda_{\min}(\Lambda)-\alpha)\tau}\mathrm{d}\tau + \int_0^t \mathrm{e}^{-\lambda_{\min}(\Lambda)(t-\tau)}\frac{B}{\alpha}\mathrm{d}\tau \tag{4.26}$$

$$\leq \mathrm{e}^{-\lambda_{\min}(\Lambda)t}\|\tilde{q}(0)\| + A\frac{\mathrm{e}^{-\alpha t} - \mathrm{e}^{-\lambda_{\min}(\Lambda)t}}{\lambda_{\min}(\Lambda) - \alpha} + \frac{B\left(1 - \mathrm{e}^{-\lambda_{\min}(\Lambda)t}\right)}{\alpha\lambda_{\min}(\Lambda)} \tag{4.27}$$

Taking the limit, we arrive at (4.25)

□

With the following corollary, we will justify that α is strictly positive even when $\phi \equiv 0$, and thus the adaptive control algorithm guarantees robustness even in the absence of persistent excitation or with ineffective learning. In practice, we expect some measurement information about all the elements of a, that is, we expect a non-zero ϕ.

Corollary 4.5.1.2. *If $\phi \equiv 0$, then the bound in (4.10) can be simplified to*

$$\lim_{t\to\infty} \left\| \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \right\| \leq \frac{\sup \|d\| + \lambda_{\max}(P^{-1}) \sup(\|\lambda a + \dot{a}\|)}{\min\left(\lambda, \lambda_{\min}(K)/\lambda_{\max}(M)\right) \lambda_{\min}(\mathcal{M})} \tag{4.28}$$

Proof. Assuming $\phi \equiv 0$ immediately leads to α of

$$\alpha = \min\left(\frac{1}{2}\lambda_{\min}(P^{-1}Q), \frac{\lambda_{\min}(K)}{\lambda_{\max}(M)}\right) \tag{4.29}$$

$\phi \equiv 0$ also simplifies the $\dot{P}$ equation to a stable first-order differential matrix equation. By integrating this simplified $\dot{P}$ equation, we can show P exponentially converges to the value $P = \frac{Q}{2\lambda}$. This leads to bound in (4.28).

□

We now introduce another corollary for the Neural-Fly-Constant, when $\phi = I$. In this case, the regularization term is not needed, as it is intended to regularize the linear coefficient estimate in the absence of persistent excitation, so we set $\lambda = 0$. This corollary also shows that Neural-Fly-Constant is sufficient for perfect tracking control when f is constant; though in this case, even the nonlinear baseline controller with integral control will converge to perfect tracking. In practice for quadrotors, we only expect f to be constant when the drone air-velocity is constant, such as in hover or steady level flight with constant wind velocity.

Corollary 4.5.1.3. *If $\phi \equiv I$, $Q = qI$, $R = rI$, $\lambda = 0$, and $P(0) = p_0 I$ is diagonal, where q, r and p_0 are strictly positive scalar constants, then the bound in (4.10) can be simplified to*

$$\lim_{t\to\infty} \left\| \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \right\| \leq \frac{\left((1 + r^{-1}) \sup_t \|f - a\| + \epsilon/r\right) \lambda_{\max}(M)}{\lambda_{\min}(K)\lambda_{\min}(\mathcal{M})} \tag{4.30}$$

Proof. Under these assumptions, the matrix differential equation for P is reduced to the scalar differential equation

$$\frac{dp}{dt} = q - p^2/r \tag{4.31}$$

where $P(t) = p(t)I$. This equation can be integrated to find that p exponentially converges to $p = \sqrt{qr}$. Then by (4.16), $\alpha \leq \sqrt{q/r}$ and $\alpha \leq \lambda_{\min}(K)/\lambda_{\max}(M)$. If we choose q and r such that $\sqrt{q/r} = \lambda_{\min}(K)/\lambda_{\max}(M)$, then we can take $\alpha = \lambda_{\min}(K)/\lambda_{\max}(M)$. Then, the error bound reduces to

$$\lim_{t\to\infty} \left\| \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \right\| \leq \frac{D\lambda_{\max}(M)}{\lambda_{\min}(K)\lambda_{\min}(\mathcal{M})} \tag{4.32}$$

Take a as a constant. Then $\dot{a} = 0$, $d = f - a$, and D is bounded by

$$D \leq \left(1 + r^{-1}\right) \sup_t \|f - a\| + \epsilon/r \tag{4.33}$$

□

4.6 Gain Tuning

The attitude controller was tuned following the method in [11]. The gains for all the position controllers tested were tuned on a step input of 1 m in the x-direction. The proportional (P) and derivative (D) gains were tuned using the baseline nonlinear controller for good rise time with minimal overshoot or oscillations. The same P and D gains were used across all methods.

The integral and adaptation gains were tuned separately for each method. In each case, the gains were increased to minimize response time until we observed having large overshoot, noticeably jittery, or oscillatory behavior. For $\mathcal{L}_1$ and INDI this gave a first-order filters with a cutoff frequency of 5 Hz. For each of the Neural-Fly methods, we used $R = rI$ and $Q = qI$, where r and q are a scalar values. The tuning method gave an R gains similar to the measurement noise of the residual force, a Q values on the order of 0.1, and λ values of 0.01.

4.7 Force Prediction Performance

The section discusses Fig. 4.3, which is useful for understanding why learning improves force prediction (which in turn improves control).

For the nonlinear baseline method, the integral (I) term compensates for the average wind effect, as seen in Fig. 4.3. Thus, the UAV trajectory remains roughly centered on the desired trajectory for all wind conditions, as seen in Fig. 3.5. The relative velocity of the drone changes too quickly for the integral-action to compensate for the changes in the wind effect. Although increasing the I gain would allow the integral control to react more quickly, a large I gain can also lead to overshoot and instability, thus the gain is effectively limited by the combined stability of the P, D, and I gains.

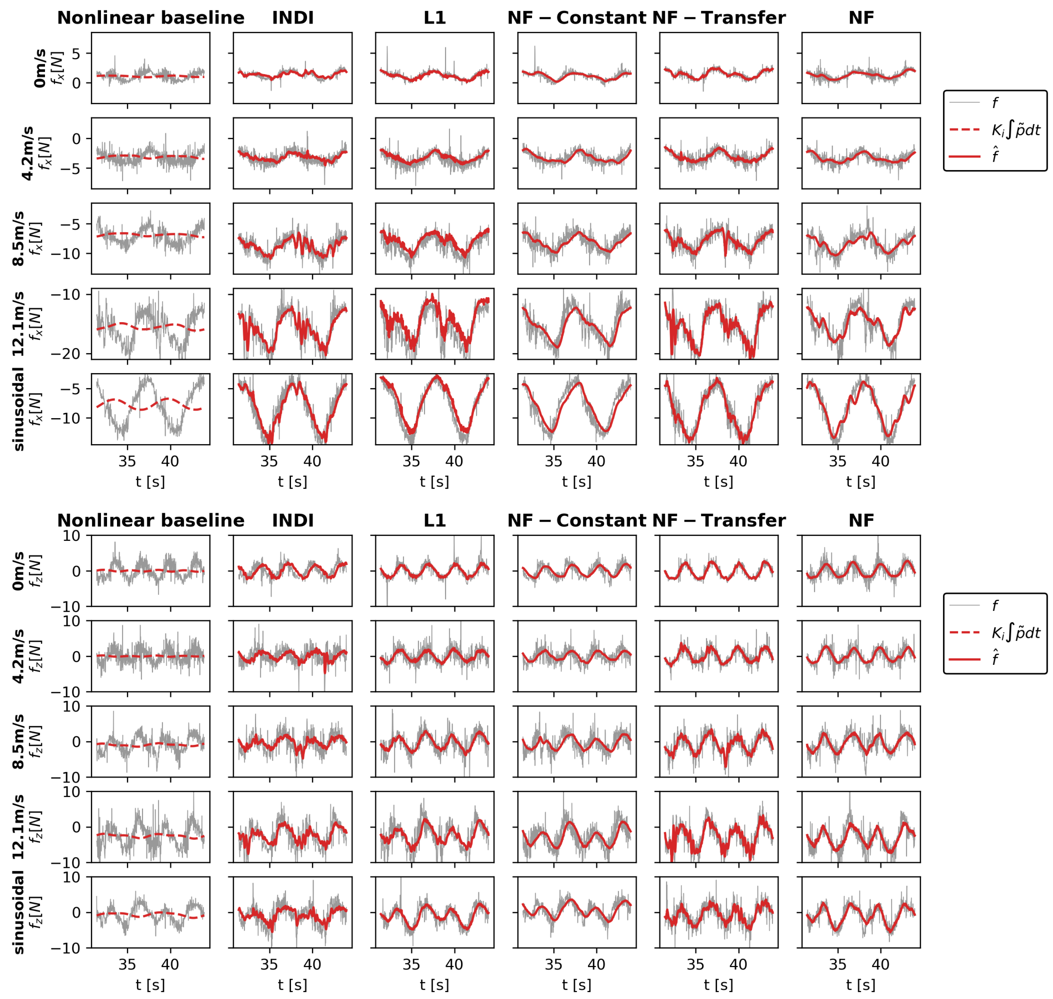


Figure 4.3: **Measured residual force versus adaptive control augmentation, $\hat{f}$.** Wind-effect x- and z-axis force prediction for different methods, $\hat{f}$ and $K_i \int \tilde{p}\mathrm{d}t$, compared with the online residual force measurement, f. The integral term in the nonlinear baseline method and the $\hat{f}$ term in the adaptive control methods, including the Neural-Fly methods, all act to compensate for the measured residual force. INDI, L1, and Neural-Fly-Constant estimate the residual force with sub-second lag, however adjusting the gains to decrease the lag increases noise amplification. Neural-Fly and Neural-Fly-Transfer have reduced the lag in estimating the residual force but have some model mismatch, especially at higher wind speeds.

Next, consider the two SOTA baseline methods, INDI and $\mathcal{L}_1$, along with the non-learning version of our method, Neural-Fly-Constant. These methods represent different adaptive control approaches that assume no prior model for the residual dynamics. Instead, each of these methods effectively outputs a filtered version of the measured residual force and the controller compensates for this adapted term. In Fig. 4.3, we observe that each of these methods has a slight lag behind the measured residual force, in gray. This lag is reduced by increasing the adaptation gain, however, increasing the adaptation gain leads to noise amplification. Thus, these *reactive* approaches are limited by some more inherent system properties, like measurement noise.

Finally, consider the two learning versions of our method, Neural-Fly and Neural-Fly-Transfer. These methods use a learned model in the adaptive control algorithm. Thus, once the linear parameters have adapted to the current wind condition, the model can predict future aerodynamic effects with minimal changes to the coefficients. As we extrapolate to higher wind speeds and time-varying conditions, some model mismatch occurs and is manifested as discrepancies between the predicted force, $\hat{f}$, and the measured force, f, as seen in Fig. 4.3. Thus, our learning-based control is limited by the learning representation error. This matches the conclusion drawn in our theoretical analysis, where tracking error scales linearly with representation error.

4.8 Localization Error Analysis

We estimate the root mean squared position localization precision to be about 1 cm. This is based on a comparison of our two different localization data sources. The first is the OptiTrack motion capture system, which uses several infrared motion tracking cameras and reflective markers on the drone to produce a delayed measurement the position and orientation of the vehicle. The PX4 flight controller runs an onboard extended Kalman filter (EKF) to fuse the OptiTrack measurements with onboard inertial measurement unit (IMU) measurements to produce position, orientation, velocity, and angular rate estimates. In offline analysis, we correct for the delay of the OptiTrack system, and compare the position outputs of the OptiTrack system and the EKF. Typical results are shown in Fig. 4.4. The fixed offset between the measurements occurs because the OptiTrack system tracks the centroid of the reflective markers, where the EKF tracks the center of mass of the vehicle. Although the EKF must internally correct for this offset, we do not need to do so in our offline analysis because the offset is fixed. Thus, the mean distance between the OptiTrack

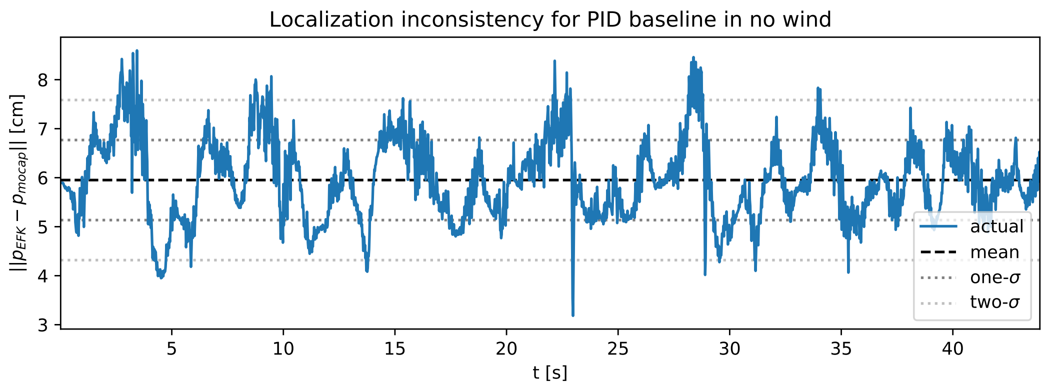


Figure 4.4: **Localization inconsistency.** Typical difference between the OptiTrack motion capture position measurement, p_{mocap}, and the EKF position estimate, p_{EKF}, corrected for the Optitrack delay. The mean difference corresponds to a constant offset between the center of mass, which the EKF tracks, and the centroid of reflective markers, which the OptiTrack measures. The standard deviation corresponds to the root-mean-square error between the two measurements.

position and the EKF position corresponds to the distance between the center of mass and the center of vision, and the standard deviation of that distance is the root-mean-square error of the error between the two estimates. Averaged over all the data from experiments in this paper, we see that the standard deviation is 1.0 cm. Thus, we estimate that the localization precision has a standard deviation of about 1.0 cm.

References

[1] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 1203–1218, May 2021, ISSN: 1558-0865. DOI: 10.1109/TCST.2020.3001117.

[2] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, Apr. 2018, ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2776353.

[3] G. Torrente, E. Kaufmann, P. Fohn, and D. Scaramuzza, "Data-driven MPC for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3769–3776, Apr. 2021, ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2021.3061307. [Online]. Available: https://ieeexplore.ieee.org/document/9361343/ (visited on 10/08/2021).

[4] L. N. Trefethen, "Multivariate polynomial approximation in the hypercube," *Proceedings of the American Mathematical Society*, vol. 145, no. 11, pp. 4837–4844, Jun. 8, 2017, ISSN: 0002-9939, 1088-6826. DOI: 10.1090/proc/

13623. [Online]. Available: https://www.ams.org/proc/2017-145-11/S0002-9939-2017-13623-5/ (visited on 03/27/2023).

[5] D. Yarotsky, "Error bounds for approximations with deep ReLU networks," *Neural Networks*, vol. 94, pp. 103–114, Oct. 1, 2017, ISSN: 0893-6080. DOI: 10.1016/j.neunet.2017.07.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608017301545 (visited on 03/27/2023).

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html (visited on 11/18/2022).

[7] L. Dieci and T. Eirola, "Positive definiteness in the numerical solution of Riccati differential equations," *Numerische Mathematik*, vol. 67, no. 3, pp. 303–313, Apr. 1, 1994, ISSN: 0945-3245. DOI: 10.1007/s002110050030. [Online]. Available: https://doi.org/10.1007/s002110050030 (visited on 09/23/2021).

[8] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1, 1960, ISSN: 0021-9223. DOI: 10.1115/1.3662552. [Online]. Available: https://doi.org/10.1115/1.3662552 (visited on 09/21/2021).

[9] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, N.J: Prentice Hall, 1991, 459 pp., ISBN: 978-0-13-040890-7.

[10] H. K. Khalil, *Nonlinear Systems, 3rd Edition*. Prentice Hall, 2002. [Online]. Available: https://www.pearson.com/content/one-dot-com/one-dot-com/us/en/higher-education/program.html (visited on 09/02/2021).

[11] "Multicopter PID tuning guide (advanced/detailed) | PX4 user guide." (), [Online]. Available: https://docs.px4.io/master/en/config_mc/pid_tuning_guide_multicopter.html (visited on 09/17/2021).

Chapter 5

FIRST ORDER DELAY COMPENSATION FOR NONLINEAR CONTROL METHODS

Abstract

Actuation delays that exist in motor dynamics and computation hardware on multirotors hinder their tracking accuracy during aggressive trajectories when not properly considered. On smaller drones, such effects are especially apparent as basic flight computers and motors cause delays that have similar timescale as vehicles' inertial dynamics. In this work, we present a light-weight augmentation method that enables trajectory tracking controllers for multirotors to handle motor dynamics and digital transport delays. We show that a simple one-step forward prediction is a first-order approximation of a conventional delay-compensating predictive controller. Using parameters from two real small quadrotors, we conduct numerical experiments to demonstrate that: (1) our proposed method achieve comparable level of trajectory tracking performance to a state-of-the-art differential flat controller when a simple regulation attitude controller is augmented; (2) the combined multi-level delay compensation method further improves the tracking accuracy becomes essential for large system delays.

5.1 Introduction

Multirotor trajectory planning and control have been extensively studied during recent years. Differential flatness property of the multirotor dynamics enables its fast and agile flights [1], even under moderate air drag [2]. The motor dynamics and computation delay in this type of vehicle has largely been ignored, due to it happening on a much faster timescale compared to the vehicle's position and attitude dynamics. For high accuracy tracking of increasing aggressive trajectories, such delays can become crucial in further improving control performance. A multirotor typically uses electric motors to drive propellers, where rotor speed or pitch angle is controlled via input signals to achieve desired thrust. In either case, delays in actuation come from digital, electrical and mechanical sources within the system. This can be especially detrimental when the delay time-scale is on the same order as the vehicle's inertial dynamics. On the other hand, with the sensitivity to total weight of flying vehicles, onboard computation is always limited compared to ground platforms. Thus digital delays due to discrete computing architecture will pose further hurdles for high performing flight control.

In general, actuation delays occur naturally in a variety of physical and cyber-physical systems. Time-delayed dynamics have been an active area of research since its introduction in 1946 [3]–[6], and it is seeing continued interest with the popularization of vast computer networks and internet-of-things (IoT) accompanied by substantial communication lags [7], [8]. Delay compensation techniques have also been widely used in control of power electronics [9], [10] and reinforcement learning settings [11]. For linear systems, delay for unstable processes is often modeled as first or second order plus dead time (FOPDT or SOPDT). Classical linear feedback control can be applied and closed-loop system behavior is analyzed with transfer function approaches. It was shown that properly designed proportional-integral-derivative (PID) controllers can act as a delay compensator [12]. Other popular techniques include relay-based identification [13] and proportional-integral-proportional-derivative (PI-PD) control [14]. For nonlinear systems, the usual consensus on the challenge of continuous delays is that the state space becomes infinite dimensional. Thus, instead of being described by ordinary differential equations (ODEs), these systems need to be modeled as functional differential equations (FDEs) or transport partial differential equations (PDEs) [5], [6]. Accordingly, their analysis requires additional mathematical tools such as Lyapunov-Krasovskii functionals [15], [16]. A prominent class of delay compensation methods rely on state predictions of some kind. This idea was first proposed as the Smith-predictor [4], and has been expanded to handle unstable processes [17], increase robustness against uncertainties [18], or adapt to varying delays [19]. In theory, predictor-based methods can handle arbitrarily large delays for forward complete and strict-feedforward systems [20].

The FDE or PDE modeling approach has the underlying assumption that input signal is continuous in time. For multirotor control systems run on digital computers, this assumption becomes less valid as command update frequency, system transport delay, and time constant for actuator dynamics exist on similar time-scales. In this work, we kept those restrictions in mind and elect to use light weight augmentations that help baseline controllers cope with various delays in the system without incurring heavy computation burden.

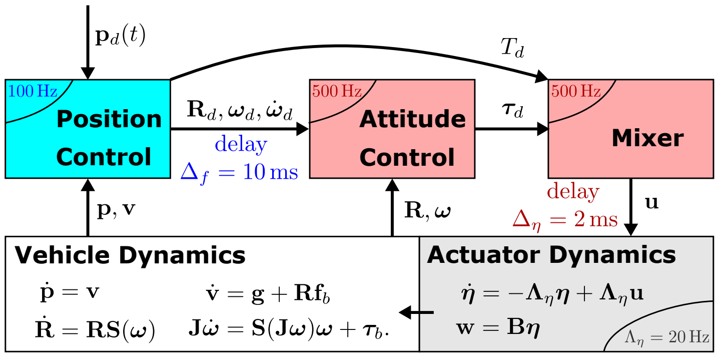

$\mathbf{p}_d(t)$
T_d
100 Hz
Position Control
$\mathbf{R}_d, \boldsymbol{\omega}_d, \dot{\boldsymbol{\omega}}_d$
delay
$\Delta_f = 10\,\mathrm{ms}$
500 Hz
Attitude Control
$\boldsymbol{\tau}_d$
500 Hz
Mixer
delay
$\Delta_\eta = 2\,\mathrm{ms}$
$\mathbf{u}$
$\mathbf{p}, \mathbf{v}$
$\mathbf{R}, \boldsymbol{\omega}$
Vehicle Dynamics
$\dot{\mathbf{p}} = \mathbf{v}$
$\dot{\mathbf{v}} = \mathbf{g} + \mathbf{R}\mathbf{f}_b$
$\dot{\mathbf{R}} = \mathbf{R}\mathbf{S}(\boldsymbol{\omega})$
$\mathbf{J}\dot{\boldsymbol{\omega}} = \mathbf{S}(\mathbf{J}\boldsymbol{\omega})\boldsymbol{\omega} + \boldsymbol{\tau}_b.$
Actuator Dynamics
$\dot{\boldsymbol{\eta}} = -\boldsymbol{\Lambda}_\eta \boldsymbol{\eta} + \boldsymbol{\Lambda}_\eta \mathbf{u}$
$\mathbf{w} = \mathbf{B}\boldsymbol{\eta}$
$\Lambda_\eta = 20\,\mathrm{Hz}$

Figure 5.1: **Hierarchy of control systems for a multirotor.** A typical drone, such as Crazyflie 2.0, has a slower outer position loop (100 Hz) than inner attitude loop (500 Hz), with even slower motor dynamics (20 Hz).

5.2 Multirotor Dynamics with Actuation Delays

We start from a conventional six degree-of-freedom (DOF) dynamics model for multirotor. The system states are defined by inertial position $\mathbf{p}$, velocity $\mathbf{v}$, attitude as rotation matrix $\mathbf{R} \in SO(3)$, and body angular rate $\boldsymbol{\omega}$. The overall equations of motion are expressed as:

$$\dot{\mathbf{p}} = \mathbf{v}, \qquad \dot{\mathbf{v}} = \mathbf{g} + \mathbf{R}\mathbf{f}_b, \tag{5.1}$$

$$\dot{\mathbf{R}} = \mathbf{R}\mathbf{S}(\boldsymbol{\omega}), \qquad \mathbf{J}\dot{\boldsymbol{\omega}} = \mathbf{S}(\mathbf{J}\boldsymbol{\omega})\boldsymbol{\omega} + \boldsymbol{\tau}_b, \tag{5.2}$$

where $\mathbf{J} \in \mathbb{R}^{3\times3}$ is the inertia matrix, $\mathbf{g}$ is the constant gravity vector in the inertial frame and $\mathbf{S}(\cdot) : \mathbb{R}^3 \to SO(3)$ is a skew-symmetric mapping such that $\mathbf{a}\times\mathbf{b} = \mathbf{S}(\mathbf{a})\mathbf{b}$. External forces and moments are split into two parts as

$$\mathbf{f}_b = \mathbf{f}_T + \mathbf{f}_A, \quad \text{and} \quad \boldsymbol{\tau}_b = \boldsymbol{\tau}_T + \boldsymbol{\tau}_A. \tag{5.3}$$

$\mathbf{f}_A$ and $\boldsymbol{\tau}_A$ are state or time dependent aerodynamic components, which can be modeled via various methods [2], [21]. $\mathbf{f}_T$ and $\boldsymbol{\tau}_T$ are generated by rotor thrusts and moments.

In this work, we only consider the flat multirotor with $n \geq 4$ propellers which can generate $\mathbf{f}_T = [T; 0; 0]$ with collective thrust $T = \sum_{i=1}^{n} T_i$ in body z-axis. Nevertheless, three-dimensional moments $\boldsymbol{\tau}_T = [\tau_{T,x}; \tau_{T,y}; \tau_{T,z}]$ are achievable. We often denote the output wrench

$$\mathbf{w} = \begin{bmatrix} T \\ \boldsymbol{\tau}_T \end{bmatrix} \in \mathbb{R}^4 \tag{5.4}$$

as a combination of collective thrust and body moments. From the physical configuration of the rotors, we can obtain a linear mapping between an output wrench $\mathbf{w}$

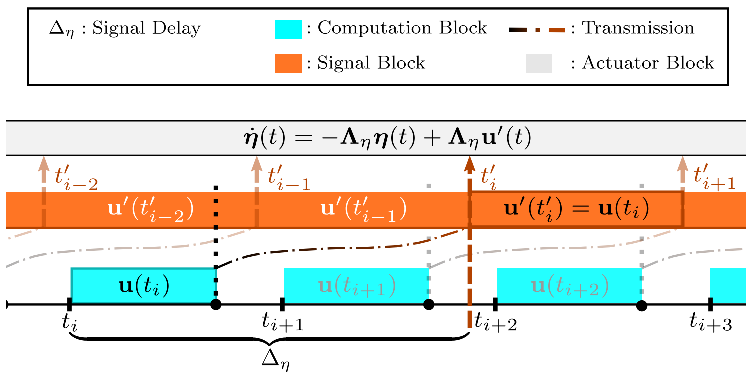


Figure 5.2: **Timeline of periodic sample-based control with discrete signal delays and actuator dynamics.** At every t_i, the computed signal $\mathbf{u}(t_i)$ takes Δ_η to reach the actuator.

and a vector of actuator forces $\boldsymbol{\eta} = [T_1; \dots; T_n]$:

$$\mathbf{w} = \mathbf{B}\boldsymbol{\eta}, \quad \mathbf{B} \in \mathbb{R}^{4\times n}. \tag{5.5}$$

Through rotor properties and bench testing, we can safely assume that $\boldsymbol{\eta}$ can be controlled through input signals $\mathbf{u} = [u_1; \dots; u_n]$. We elect to use a sample-based first-order plus dead time (SBFOPDT) model to represent the discrete and the continuous delays in the actuation system:

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= -\boldsymbol{\Lambda}_\eta \boldsymbol{\eta} + \boldsymbol{\Lambda}_\eta \mathbf{u}', \quad \mathbf{u}'(t) = \mathbf{u}(t_i), \\ \text{with} &\quad t \in [t'_i, t'_{i+1}), \quad \boldsymbol{\Lambda}_\eta = \operatorname{diag}(\lambda_{\eta_1}, \dots, \lambda_{\eta_n}). \end{aligned} \tag{5.6}$$

$\boldsymbol{\Lambda}_\eta$ is a positive definite diagonal matrix whose entries are the first order gains for motors. It typically indicates how fast the propellers spin-up to the target speed. As illustrated in Fig. 5.2, the signal $\mathbf{u}(t_i)$ is computed at a periodic sampling time t_i then delayed by Δ_η when it is received by the actuator as $\mathbf{u}'(t'_i)$ and held constant for the duration of the period between $[t'_i, t'_{i+1})$.

5.3 Delay Compensation Control

Multirotor Position and Attitude Controllers

Multirotor controllers often hierarchically consist of outer loop position controller, and inner loop attitude controller. Before addressing the actuation delay issue, we will start from a set of nonlinear feedback controllers similar to the ones used in [22]. Along desired position $\mathbf{p}_d(t)$ and attitude $\mathbf{R}_d(t)$ trajectories, we can define a reference velocity and a reference angular rate as

$$\mathbf{v}_r = \dot{\mathbf{p}}_d - \mathbf{K}_p \tilde{\mathbf{p}}, \tag{5.7}$$

$$\omega_r = \tilde{\mathbf{R}}^\top \omega_d - \mathbf{K}_q \tilde{\mathbf{q}}_v. \tag{5.8}$$

The position tracking error is $\tilde{\mathbf{p}} = \mathbf{p} - \mathbf{p}_d$. The attitude tracking error is represented via a error rotation matrix $\tilde{\mathbf{R}} = \mathbf{R}_d^\top \mathbf{R}$ and a error quaternion $\tilde{\mathbf{q}} = [\tilde{q}_0; \tilde{\mathbf{q}}_v]$. $\mathbf{K}_p$ and $\mathbf{K}_q$ are positive definite gain matrices. Thus we can drive desired force and torque as

$$\bar{\mathbf{f}} = -\mathbf{g} + \dot{\mathbf{v}}_r - \mathbf{K}_v(\mathbf{v} - \mathbf{v}_r), \tag{5.9}$$

$$\bar{\boldsymbol{\tau}} = \mathbf{J}\dot{\boldsymbol{\omega}}_r - \mathbf{S}(\mathbf{J}\boldsymbol{\omega})\boldsymbol{\omega}_r - \mathbf{K}_\omega(\boldsymbol{\omega} - \boldsymbol{\omega}_r) \tag{5.10}$$

in order to achieve exponential convergence of a desired position and attitude. Similarly, $\mathbf{K}_v$ and $\mathbf{K}_\omega$ are also positive definite gain matrices. For better trajectory tracking performance, we can also analytically derive the derivatives of $\mathbf{v}_r$ and $\boldsymbol{\omega}_r$. $\dot{\mathbf{v}}_r$ is simple differentiation from (5.7)

$$\dot{\mathbf{v}}_r = \ddot{\mathbf{p}}_d - \mathbf{K}_p\tilde{\mathbf{p}}. \tag{5.11}$$

$\dot{\boldsymbol{\omega}}_r$ is more involved, as we need to take the derivative of rotation matrix $\tilde{\mathbf{R}}$. After algebraic simplification, we can get

$$\begin{aligned}\dot{\boldsymbol{\omega}}_r = {} & \tilde{\mathbf{R}}^\top\dot{\boldsymbol{\omega}}_d - \mathbf{S}(\boldsymbol{\omega})\tilde{\mathbf{R}}^\top\boldsymbol{\omega}_d \\ & - \frac{1}{2}\mathbf{K}_q\left(\tilde{q}_0\mathbf{I} + \mathbf{S}(\tilde{\mathbf{q}}_v)\right)(\boldsymbol{\omega} - \tilde{\mathbf{R}}^\top\boldsymbol{\omega}_d).\end{aligned} \tag{5.12}$$

Multirotor dynamics have been proved to be differential flat, thus methods from [1] can be applied to obtain quantities $\boldsymbol{\omega}_d$ and $\dot{\boldsymbol{\omega}}_d$ from a 5th-order differentiable trajectory $\mathbf{p}_d(t)$.

Delay Compensation at Position Control Level

A persistent issue that comes with multirotor hierarchical controller is the determination of attitude, i.e., solving $\mathbf{R}_d\mathbf{f}_b = \bar{\mathbf{f}}$. Based on the desired force $\bar{\mathbf{f}}$ from (5.9), a typical selection of the desired z-axis of the vehicle without considering aerodynamics disturbance $\mathbf{f}_A$ is $\hat{\mathbf{z}}_d = \bar{\mathbf{f}}/\|\bar{\mathbf{f}}\|$, which consequently determines the full desired attitude $\mathbf{R}_d$ given yaw angle command. Although there are extensions of this simple method that explicit take into account drag forces [2] or general aerodynamic forces [22], they all disregard the time needed for the attitude error to converge.

From (5.2), a high-order dynamics of $\dot{\bar{\mathbf{f}}}$ can be constructed and compensated for during control design; however, doing so inevitably requires information of acceleration and jerk of the vehicle. Instead, we propose a simplified substitute for this high-order dynamics with a FOPDT model:

$$\begin{aligned}\dot{\mathbf{f}}_b &= -\boldsymbol{\Lambda}_f\mathbf{f}_b + \boldsymbol{\Lambda}_f\mathbf{f}_d(t - \Delta_f), \\ \text{with} \quad \boldsymbol{\Lambda}_f &= \operatorname{diag}(\lambda_{xy}, \lambda_{xy}, \lambda_z).\end{aligned} \tag{5.13}$$

This model approximates the high-order dynamics using first-order delays. Since fb is expressed in the body frame, its delays are represented in two parts: (1) λ_{xy} denotes the delay in attitude convergence; (2) λ_z denotes the delay from T in body z-axis. The model also take into account discrete transport delay Δ_f internal to the hardware. In practice, Δ_f includes the time needed for position controller to finish computing.

Theorem 5.3.1. *The following delay compensation controller*

$$\mathbf{f}_d = \bar{\mathbf{f}} + \left(\mathbf{\Lambda}_f^{-1} + \Delta_f\right)\dot{\bar{\mathbf{f}}} \tag{5.14}$$

derived from the desired force computed in the position controller (5.9), exponentially stabilizes the FOPDT system (5.13) $\mathbf{f}_b \to \bar{\mathbf{f}}$ *when truncated to first-order.*

Proof. We can approximate (5.13) to first-order as

$$\dot{\mathbf{f}}_b = -\mathbf{\Lambda}_f\mathbf{f}_b + \mathbf{\Lambda}_f\mathbf{f}_d - \mathbf{\Lambda}_f\Delta_f\dot{\mathbf{f}}_d + O(\Delta_f^2).$$

By defining a force error $\mathbf{e}_f = \mathbf{f}_b - \bar{\mathbf{f}}$ and substituting (5.14) into the above approximation, we can write the error dynamics of $\mathbf{e}_f$ as

$$\begin{aligned}\dot{\mathbf{e}}_f &= -\mathbf{\Lambda}_f\mathbf{e}_f - \mathbf{\Lambda}_f\Delta_f\left(\mathbf{\Lambda}_f^{-1} + \Delta_f\right)\ddot{\mathbf{f}}_d + O(\Delta_f^2)\\ &= -\mathbf{\Lambda}_f\mathbf{e}_f + O(\ddot{\mathbf{f}}_d) + O(\Delta_f^2).\end{aligned}$$

When neglecting the higher-order terms of Δ_f^2 and $\ddot{\mathbf{f}}_d$, it can be shown that $\mathbf{e}_f \to 0$ exponentially with rate $\lambda_{\min}(\mathbf{\Lambda}_f)$. □

From the hierarchy illustrated in Fig. 5.1, Theorem 5.3.1 states a method to compensate for delays in inner loop controllers by adjusting desired force $\mathbf{f}_d$ according to (5.14). Without loss of generality, we neglect aerodynamics disturbance $\mathbf{f}_A$ when determining desired collective thrust T_d and attitude $\mathbf{R}_d$:

$$T_d = \mathbf{f}_d \cdot \hat{\mathbf{z}}, \quad \text{and} \quad \hat{\mathbf{z}}_d = \mathbf{f}_d/\|\mathbf{f}_d\|, \tag{5.15}$$

where $\hat{\mathbf{z}}$ and $\hat{\mathbf{z}}_d$ are the unit vectors for vehicle's current and desired z-axis. It is straightforward to compute desired attitude $\mathbf{R}_d$ from $\hat{\mathbf{z}}_d$ when desired yaw angle is also specified [1]. Methods from [2], [22] can also be used in place of (5.15) without affecting derivations of further compensations of delays in the following section.

Delay Compensation at Mixer Level

Combining (5.10) and (5.15) from the position and attitude controllers, we assemble the desired wrench vector $\mathbf{w}_d$ to be

$$\mathbf{w}_d = \begin{bmatrix} T_d \\ \boldsymbol{\tau}_d \end{bmatrix} \in \mathbb{R}^4. \tag{5.16}$$

Similarly as before, we neglect aerodynamic moments $\boldsymbol{\tau}_A$ and choose $\boldsymbol{\tau}_d = \bar{\boldsymbol{\tau}}$ for simplicity. We denote the process that calculate $\mathbf{u}$ from $\mathbf{w}_d$ as the mixer. Sometimes it is also referred as control allocation. Figure 5.1 shows its location within the control stack. We are ready to state a delay compensated mixer in the following theorem.

Theorem 5.3.2. *Considering the actuator FOPDT delay model (5.6) and a control mapping (5.5), the following mixer with delay compensation*

$$\mathbf{u} = \mathbf{B}^{-1}\mathbf{w}_d + \left(\boldsymbol{\Lambda}_\eta^{-1} + \Delta_\eta\right)\mathbf{B}^{-1}\dot{\mathbf{w}}_d \tag{5.17}$$

exponentially stabilizes the wrench error $\mathbf{w} \to \mathbf{w}_d$ *when truncated to first-order.*

Proof. Similar to the proof given in Theorem 5.3.1, we can approximate (5.6) to first-order as

$$\dot{\boldsymbol{\eta}} = -\boldsymbol{\Lambda}_\eta\boldsymbol{\eta} + \boldsymbol{\Lambda}_\eta\mathbf{u} - \boldsymbol{\Lambda}_\eta\Delta_\eta\dot{\mathbf{u}} + \mathcal{O}(\Delta_\eta^2).$$

Let wrench error be $\mathbf{e}_w = \mathbf{w} - \mathbf{w}_d$. When substituting (5.17) and the above approximation into (5.5), the error dynamics can be written as

$$\begin{aligned}\dot{\mathbf{e}}_w &= \mathbf{B}\dot{\boldsymbol{\eta}} - \dot{\mathbf{w}}_d \\ &= -\mathbf{B}\boldsymbol{\Lambda}_\eta\mathbf{B}^{-1}\mathbf{e}_w + \dot{\mathbf{w}}_d + \mathbf{B}\boldsymbol{\Lambda}_\eta\Delta_\eta\mathbf{B}^{-1}\dot{\mathbf{w}}_d - \mathbf{B}\boldsymbol{\Lambda}_\eta\Delta_\eta\dot{\mathbf{u}} \\ &\quad - \dot{\mathbf{w}}_d + \mathcal{O}(\Delta_\eta^2) \\ &= -\mathbf{B}\boldsymbol{\Lambda}_\eta\mathbf{B}^{-1}\mathbf{e}_w + \mathcal{O}(\ddot{\mathbf{w}}_d) + \mathcal{O}(\Delta_\eta^2).\end{aligned}$$

When neglecting the higher order terms of Δ_η^2 and $\ddot{\mathbf{w}}_d$, it can be shown that $\mathbf{e}_w \to 0$ exponentially with rate $\lambda_{\min}(\boldsymbol{\Lambda}_\eta)$. □

At this stage, we augment a baseline position and attitude controller such as the ones from (5.9) and (5.10) with delay compensation techniques. It accounts for delays at both the position control level as well as the mixer level with tunable parameters $\boldsymbol{\Lambda}_f$, Δ_f, $\boldsymbol{\Lambda}_\eta$, and Δ_η.

Table 5.1: **Simulation parameters for two drones of different sizes.**

		Intel Aero	Crazyflie 2.0
mass	m	2.40 kg	0.033 kg
inertia	J_x	3.60×10^{-3} kg m^2	1.57×10^{-5} kg m^2
	J_y	3.60×10^{-3} kg m^2	1.66×10^{-5} kg m^2
	J_z	7.20×10^{-3} kg m^2	2.93×10^{-5} kg m^2
arm length		30 cm	4.6 cm
thrust-to-weight		1.43	1.41
pos ctrl freq		50 Hz	100 Hz
pos ctrl delay	Δ_f	20 ms	10 ms
att ctrl freq		250 Hz	500 Hz
ang ctrl freq		500 Hz	500 Hz
motor gain	$\boldsymbol{\Lambda}_\eta$	50 Hz	20 Hz
mixer delay	Δ_η	2 ms	2 ms

5.4 Numerical Experiments

Simulation Setups

In this section, we use numerical simulations to study the effects that delay compensation techniques at different levels have on the trajectory tracking performance of two quadrotor drones based on real world hardware settings. The two drones of interest are the Intel Aero Ready to Fly Drone and the Crazyflie 2.0. Table 5.1 lists various physical properties as well as computation parameters of the two platforms.

The two quadrotor platforms are 10 times different in size and 100 times different in mass and inertia, yet the control-related delays and time-scales are on the same order of magnitude.

Definition of Different Controllers for Comparisons

In order to see the benefits of different delay compensation techniques, we select several controllers to compare their performances on different trajectories. The details about each of them are introduced below with a summary listed in Table 5.2.

Cascaded PID

First, we show that our method can extend a common cascaded PID controller for quadrotors. The position control is similar to (5.9) with a PID feedback component and a feedforward component:

$$\mathbf{f}_d = -\mathbf{g} + \ddot{\mathbf{p}}_d - \mathbf{K}_P\tilde{\mathbf{p}} - \mathbf{K}_D\dot{\tilde{\mathbf{p}}} - \mathbf{K}_I \int \tilde{\mathbf{p}} dt. \tag{5.18}$$

The attitude determination is the same as in (5.15). And the attitude controller is a regulation PID feedback on ω:

$$\begin{aligned} \omega_r &= -\mathbf{K}_q \tilde{\mathbf{q}}_v, \quad \tilde{\omega} = \omega - \omega_r, \\ \boldsymbol{\tau}_d &= -\mathbf{K}_{P,\omega}\tilde{\omega} - \mathbf{K}_{D,\omega}\dot{\tilde{\omega}} - \mathbf{K}_{I,\omega} \int \tilde{\omega} dt. \end{aligned} \tag{5.19}$$

The mixer without delay compensation technique is simply

$$\mathbf{u} = \mathbf{B}^{-1}\mathbf{w}_d. \tag{5.20}$$

Common commercial flight controllers such as BetaFlight and PX4 [23] deploy similar attitude control laws and mixers as (5.19) and (5.20).

Position Level Delay Compensation

Next, we use the position controller (5.9), the associated delay compensation (5.14) described in Sec. 5.3, and the regulation PID (5.19) for attitude control described above to showcase high-level compensation can drastically improve performance even low-level attitude control is only of a regulation type.

Differential Flatness

We can use nonlinear attitude controller described in (5.8) and (5.10) to achieve better tracking performance if desired angular rate $\boldsymbol{\omega}_d$ and angular acceleration $\dot{\boldsymbol{\omega}}_d$ along our trajectory $\mathbf{p}_d(t)$ is calculated using (5.11) and (5.12) and differential flatness properties of multirotor dynamics. This formulation has been vastly used in aggressive trajectory tracking of multirotor drones to achieve state-the-art performances.

Mixer Level Delay Compensation

Built on top of differential flatness controller above, the actuation delays of the motors can be compensated using (5.17).

Table 5.2: **Summary of controllers definitions.**

Abbrev. Name	Position Control	Attitude Control	Mixer
Casc. PID	(5.15) and (5.18)	(5.19)	(5.20)
Pos. DC	(5.9), (5.14) and (5.15)	(5.19)	(5.20)
Diff. Flat.	(5.9) and (5.15)	(5.10)	(5.20)
Mixer DC	(5.9) and (5.15)	(5.10)	(5.17)
Comb. DC	(5.9), (5.14) and (5.15)	(5.10)	(5.17)

Combined Delay Compensation

Finally, the combined method where both the position level and the mixer level delay compensation are applied to differential flatness controller are included as the theoretical performance ceiling of the techniques proposed in this work.

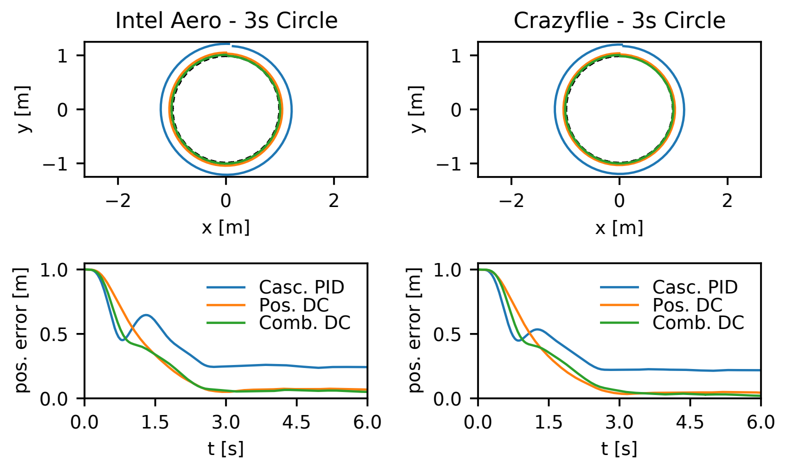
Comparisons of Trajectory Tracking Performance

From Fig. 5.3 and Table 5.3, we see that all the control methods except the Cascaded PID approach perform well tracking the xy-plane circular trajectory, with the combined delay compensation method performing best by a small margin. This is primarily because the desired thrust along the circle is constant, so methods that compensate for thrust delays do not offer an advantage on this particular trajectory. The Intel Aero drone, which runs the position control on a separate flight computer, runs the position control loop at a slower frequency. This is one of the main reasons for degraded performance compared to the Crazyflie platform, which runs the position control on the flight controller.

In contrast to the circular trajectory, the figure-∞ trajectory has a significant variation in thrust along the desired trajectory. Consequently, in Fig. 5.4 and Table 5.3, we observe the performance improvement the delay compensation approaches we propose, which explicitly account for thrust delay and other system delays. Particularly surprising is that the Position Compensation approach for the Crazyflie platform outperforms the differential flatness control with a significantly simpler attitude controller.

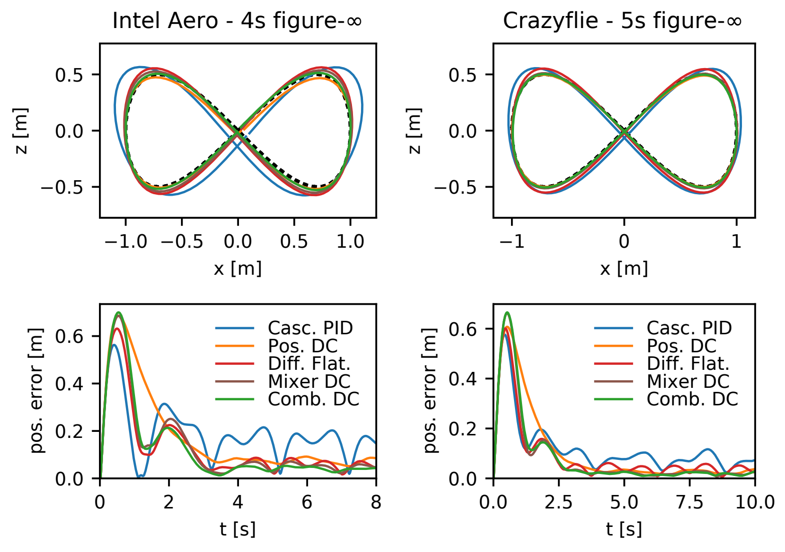
Combined Delay Compensation Technique on System with High Delay

As Fig. 5.3 and 5.4 as well as Table 5.3 show that the delay compensation at both position control and mixer levels improve tracking performance over baseline methods. The combined delay compensation has about 10% reduction in error for circu-



(a) Intel Aero 3 s Circle (b) Crazyflie 2.0 3 s Circle

Figure 5.3: **Simulation results tracking a fast circle trajectory.** Intel Aero and Crazyflie 2.0 simulation tracking a 2 m wide circle in the xy-plane with a period of 3 s, starting from hover at the origin. Top plots show steady state trajectory, bottom plots show last 6 s of flights.



(a) Intel Aero 4 s Figure-∞ (b) Crazyflie 2.0 5 s Figure-∞

Figure 5.4: **Simulation results tracking a fast Figure-∞ trajectory.** Intel Aero and Crazyflie 2.0 simulation tracking a Figure-∞ in the xz-plane, starting from hover at the origin. The periods are 4 s and 5 s, respectively. Top plots show steady state trajectory, bottom plots show last 6 s of flights.

Table 5.3: **Summary of trajectory tracking performance.**

Intel Aero		
Trajectory	Controller	$\|\|\tilde{\mathbf{p}}\|\|_{\text{RMS}}$ [cm]
Circle (3 s)	Casc. PID	24.5
	Pos. DC	7.10
	Diff. Flat.	6.88
	Mixer DC	6.91
	Comb. DC	6.04
Figure-∞ (4 s)	Cascaded PID	16.3
	Pos. DC	7.29
	Diff. Flat.	6.44
	Mixer DC	5.51
	Comb. DC	4.41
Crazyflie 2.0		
Circle (3 s)	Cascaded PID	21.9
	Pos. DC	4.58
	Diff. Flat.	3.42
	Mixer DC	3.47
	Comb. DC	3.04
Figure-∞ (5 s)	Cascaded PID	8.31
	Pos. DC	3.07
	Diff. Flat.	4.46
	Mixer DC	2.64
	Comb. DC	2.23

lar trajectories in both drones, and 50% reduction for Figure-∞ trajectories. Such observation motivates us to investigate further its capabilities when trajectories become more aggressive, or similarly during high delay situations. We conducted the same trajectory tracking task as in Fig. 5.4 for Intel Aero comparing Diff. Flat. and Comb. DC method, at the same time varied motor delay and signal delay values for the simulations. As shown in Fig. 5.5, a drastic improvement of delay compensation controllers in RMS error can be observed for both high motor and/or high signal delays. A Diff. Flat. controller struggles as signal delays are over 100 ms. It also indicates that the proposed method handles the mixed delay sources with ease, as combination of continuous and discrete delays persist in these simulations.

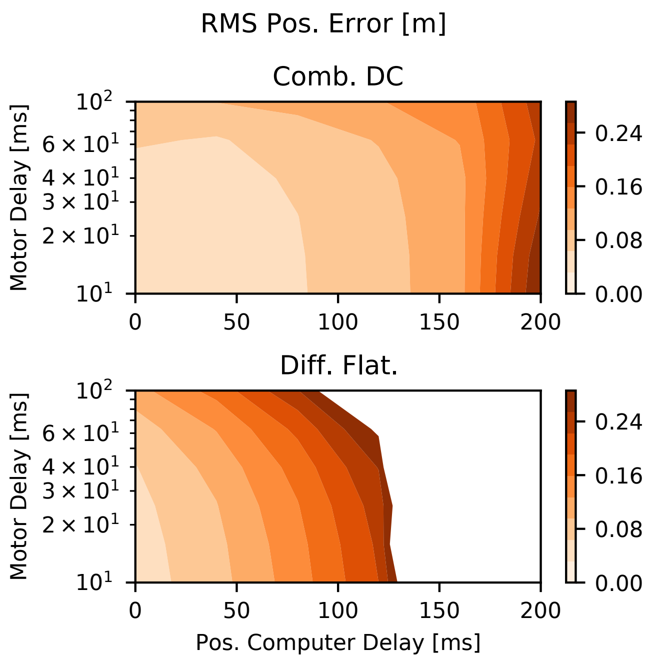


Figure 5.5: **Heatmap showing performance degradation with increasing delays for the proposed delay compensation controller and the baseline differential flatness controller.** Heat map plots of RMS position tracking error, in meters, versus the position control loop computer delay and motor delay for the Intel Aero simulation tracking the 4 s figure-∞ in Fig. 5.4. Top: proposed delay compensation control scheme. Bottom: baseline nonlinear controller based on differential flatness.

5.5 Conclusion

We proposed simple augmentation techniques for different levels of multirotor trajectory tracking controllers. We utilized first-order approximations as surrogate models for high-order attitude dynamics and designed a controller that compensate for force delays that stem from lags in attitude and collective thrust command. When applying differential flat control to trajectory tracking tasks, we proposed further upgrades by deploying mixer level delay compensation that accounts for motor dynamics. Both methods consider not only the continuous dynamics of the multirotor, but also discrete signal delays due to the computation hardware. When comparing to existing methods on realistic simulations, a position level strategy can achieve comparable result to differential flat control even though the former used only regulation control in attitude. The combined delay compensation techniques showed further improvement to achieve the best results overall, and was proved essential when system delays are further increased. Overall, our method is simple yet highly effective. The real world motivated simulation settings pave ways for the application of our proposal on real hardware experiments in the future.

References

[1] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[2] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2017.

[3] Y. Z. Tsypkin, "The systems with delayed feedback," *Avtomathika i Telemech*, vol. 7, pp. 107–129, 1946.

[4] O. J. Smith, "A controller to overcome dead time," *ISA Journal*, vol. 6, pp. 28–33, 1959.

[5] J.-P. Richard, "Time-delay systems: An overview of some recent advances and open problems," *Automatica*, vol. 39, no. 10, pp. 1667–1694, 2003.

[6] M. Krstic, *Delay Compensation for Nonlinear, Adaptive, and PDE Systems*. Springer, 2009.

[7] H. Gao, T. Chen, and J. Lam, "A new delay system approach to network-based control," *Automatica*, vol. 44, no. 1, pp. 39–52, 2008.

[8] R. A. Gupta and M.-Y. Chow, "Networked control system: Overview and research trends," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 7, pp. 2527–2535, 2009.

[9] P. Cortes, J. Rodriguez, C. Silva, and A. Flores, "Delay compensation in model predictive current control of a three-phase inverter," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 2, pp. 1323–1325, 2011.

[10] M. Lu, X. Wang, P. C. Loh, F. Blaabjerg, and T. Dragicevic, "Graphical evaluation of time-delay compensation techniques for digitally controlled converters," *IEEE Transactions on Power Electronics*, vol. 33, no. 3, pp. 2601–2614, 2017.

[11] E. Schuitema, L. Buşoniu, R. Babuška, and P. Jonker, "Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 3226–3231.

[12] A. Visioli, *Practical PID Control*. Springer Science & Business Media, 2006.

[13] P. K. Padhy and S. Majhi, "Relay based PI–PD design for stable and unstable FOPDT processes," *Computers & Chemical Engineering*, vol. 30, no. 5, pp. 790–796, 2006.

[14] S. Majhi and D. Atherton, "Online tuning of controllers for an unstable FOPDT process," *IEE Proceedings-Control Theory and Applications*, vol. 147, no. 4, pp. 421–427, 2000.

[15] V. L. Kharitonov and A. P. Zhabko, "Lyapunov–Krasovskii approach to the robust stability analysis of time-delay systems," *Automatica*, vol. 39, no. 1, pp. 15–20, 2003.

[16] F. Mazenc, S.-I. Niculescu, and M. Krstic, "Lyapunov–Krasovskii functionals and application to input delay compensation for linear time-invariant systems," *Automatica*, vol. 48, no. 7, pp. 1317–1323, 2012.

[17] M. A. Henson and D. E. Seborg, "Time delay compensation for nonlinear processes," *Industrial & Engineering Chemistry Research*, vol. 33, no. 6, pp. 1493–1500, 1994.

[18] Y.-H. Roh and J.-H. Oh, "Robust stabilization of uncertain input-delay systems by sliding mode control with delay compensation," *Automatica*, vol. 35, no. 11, pp. 1861–1865, 1999.

[19] D. Bresch-Pietri and M. Krstic, "Adaptive trajectory tracking despite unknown input delay and plant parameters," *Automatica*, vol. 45, no. 9, pp. 2074–2081, 2009.

[20] M. Krstic, "Input delay compensation for forward complete and strict-feedforward nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 2, pp. 287–303, 2009.

[21] G. Shi, X. Shi, M. O'Connell, *et al.*, "Neural lander: Stable drone landing control using learned dynamics," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 9784–9790. DOI: `10.1109/ICRA.2019.8794351`,

[22] X. Shi, K. Kim, S. Rahili, and S.-J. Chung, "Nonlinear control of autonomous flying cars with wings and distributed electric propulsion," in *2018 IEEE Conference on Decision and Control (CDC)*, Miami Beach, FL: IEEE, Dec. 2018, pp. 5326–5333, ISBN: 978-1-5386-1395-5. DOI: `10.1109/CDC.2018.8619578`. [Online]. Available: `https://ieeexplore.ieee.org/document/8619578/` (visited on 08/25/2020).

[23] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "PIXHAWK -A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1-2, pp. 21–39, Aug. 2012, ISSN: 0929-5593. DOI: `10.1007/s10514-012-9281-4`. [Online]. Available: `https://graz.pure.elsevier.com/en/publications/pixhawk-a-micro-aerial-vehicle-design-for-autonomous-flight-using` (visited on 08/31/2021).

Chapter 6

NEURAL FLY FOR FAULT TOLERANCE

Abstract

Unmanned Aerial Vehicles (UAVs) are increasingly employed in various applications, emphasizing the need for robust fault detection and compensation strategies to ensure safe and reliable operation. This study presents a novel sparse failure identification method for detecting and compensating for motor failures in octorotor UAVs. This algorithm is an extension of the learning-based control framework, Neural-Fly. The proposed method leverages a reformulation of the Neural-Fly online adaptation algorithm and a unique allocation update approach to prevent saturation and improve tracking performance in the presence of modeling errors and actuator faults. Preliminary results demonstrate the ability to isolate a single motor failure within one second and rebalance the system, with the UAV remaining controllable and experiencing minimal performance degradation. The proposed allocation update approach outperforms existing methods, improving maximum torque without saturation at hover by more than 50% compared to the pseudoinverse method. When direct motor speed sensing is available, the proposed allocation algorithm and control architecture enables almost instantaneous system correction. While the method shows promise in handling single motor failures, its performance in the presence of multiple motor failures requires further investigation. Future work will focus on addressing this limitation, comparing the proposed method with other adaptive control algorithms, and refining efficiency estimation techniques. The findings of this study contribute to the development of robust fault detection and compensation strategies for UAVs, enhancing their safety and reliability in a wide range of applications.

6.1 Introduction

Uninhabited aerial vehicles (UAVs) carry the potential to revolutionize a diverse range of industries. From emergency response services to last-mile delivery, the applications are vast and continually expanding. However, the promise of this technology comes with significant challenges. Stricter safety requirements, especially for operation in populated areas, demand UAVs that can tolerate a wide array of faults. The cost of failure in this context isn't just financial, it's potentially catastrophic, involving risks to bystanders and infrastructure. As such, UAV systems must maintain a high safety and reliability margin during all phases of operation.

Operation of UAVs typically includes several modes such as vertical take-off and landing, hover and approach phases, and a cruise stage. Transitioning smoothly and safely between these different modes, often in highly dynamic environments, is a complex and demanding task. Further complicating the matter, many next-generation aircraft prototypes feature highly redundant control actuators. While this redundancy aims to ensure sufficient control authority to recover in the event of a failure, it also increases the complexity of the control system. Standard practice is to evaluate the system through a fault tree analysis, considering all foreseeable failure scenarios. However, the potential for unseen failures becomes more likely with growing system complexity. This poses a significant challenge: how to design and implement reliable fault-tolerant control systems capable of handling this complexity and diversity of potential failure scenarios.

The existing body of research commonly dissects the complex problem of recovering from faults into discrete steps, with a primary focus either on fault diagnosis—which includes detection, isolation, and identification—or on fault-tolerant control. Recent surveys, such as those by [1] and [2], provide comprehensive overviews of various fault diagnosis methodologies. Importantly, a fault-tolerant architecture must extend beyond diagnosis, possessing the capability to adjust the control system in response to identified faults.

On the other hand, a considerable portion of work on fault-tolerant control (FTC) concentrates on recovery from faults without taking into account the steps for fault diagnosis. Control allocation algorithms emerge as particularly useful in this context [3]–[7]. These algorithms serve dual purposes: they are tools for designing robust vehicle configurations [4], [5], and they can be used for on-the-fly to response to identified faults [3]–[5]. In a related vein, [8] presents methods for analyzing vehicle performance limits under specific, enumerated failure scenarios.

Lastly, there is a substantial body of work that addresses both fault diagnosis and fault-tolerant control concurrently. Early studies in the field, such as [9], [10], utilized the Interacting Multiple Model approach. This method propagates a likelihood of enumerated failure cases and then uses that likelihood to fuse the ideal estimated state or control command for each isolated failure. More recent work focuses on isolating the most likely fault, and then performs discrete switching between nominal and fault-compensating control schemes [11]–[13]. Certain studies, like [14], build upon adaptive control tools, but are limited by the need for persistent excitation of the failure detection signal. [15] addresses this limitation by recording key previous measurements, while [16] uses both output tracking performance and direct sensor measurements to design robust control schemes.

Following this line of thought, it's important to note that many robust control algorithms and adaptive control algorithms, originally developed for other contexts, can be applied effectively to fault-tolerant control. This is especially true in the case of benign failures, as illustrated by the methods presented in [17]–[23]. These strategies deliver impressive and agile performance under nominal conditions. For highly over-actuated vehicles, these methods prove to be an appropriate solution for handling benign faults. However, their effectiveness dwindles in severe cases where the controllability of the system is dramatically reduced or altered.

While numerous methodologies exist for fault detection, isolation, and fault-tolerant control, there is a significant gap in addressing and correcting for unforeseen failures, which were not considered during the design of the fault mitigation system. In response to this challenge, we propose a method applicable to a wide range of over-actuated systems. The proposed approach caters to faults that are not directly sensed, but are inferred from the aircraft's response to control commands. Additionally, our method is designed to recover the original performance of the system, enabling safe recovery from any operational mode. Building upon prior work in [17], we incorporate an online-learning-based scheme, which facilitates precise and agile control even amidst faults.

The first significant contribution of this work is a sparse-fault-identification extension to the online learning control system, Neural-Fly [17]. The second contribution is a novel control allocation scheme that effectively leverages both the learned dynamics and failure identification. This scheme redesigns the control allocation matrix to maintain the original system performance and maximize control authority. Lastly, we demonstrate our method's effectiveness through simulated and real-world tests on an octorotor, providing a practical and compelling example of our approach.

6.2 Preliminaries

System Dynamics

Consider dynamics of the form

$$\dot{x} = f(x) + B(t)u + f_{\text{res}}(x,u,t) + d(t). \tag{6.1}$$

where $x \in \mathbb{R}^n$ is the n-dimensional state vector, $f(x)$ is a known nominal dynamics model, $B(t)$ is an unknown control actuation matrix that is approximated by $B_0 \approx B(t)$ when the system operates nominally, $u \in \mathbb{R}^m$ is the m-dimensional control input, $f_{\text{res}}(x,u,t)$ incorporates unknown residual dynamics, and $d(t)$ is random noise. Without loss of generality, assume that $u \in [0,1]^m$ and $\tau_{\max,i} = \max_u |(B_0 u)_i|$.

For small enough g, the system is exponentially stabilized by the feedback linearization control law,

$$u_n = {B_0}_R^{-1}\tau_d, \qquad \tau_d = (-K(x - x_d) + \dot{x}_d - f(x)), \tag{6.2}$$

where K is a positive definite gain matrix, x_d and $\dot{x}_d$ are the desired state and derivative of the desired state, respectively, and ${B_0}_R^{-1}$ is any right pseudoinverse of B_0 such that $B_0 {B_0}_R^{-1} = \mathbb{I}$.

We consider actuator faults of the form

$$B(t) = BH(t) \tag{6.3}$$

$$H(t) = \begin{bmatrix} \eta_1(t) & 0 & \dots & 0 \\ 0 & \eta_2(t) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \eta_m(t) \end{bmatrix} \tag{6.4}$$

where $\eta_i(t) \in [0,1]$ is the efficiency factor for the ith actuator at time t.

Assume that we have a noisy measurement, y, of $g(x,u,t)$, the error in the nominal model,

$$\begin{aligned} y &= g(x,u,t) + \epsilon \\ g(x,u,t) = \dot{x} - f(x) - B_0 u &= f_{\text{res}}(x,u,t) + B_0(\mathbb{I} - H(t))u, \end{aligned} \tag{6.5}$$

where $\epsilon(t) > 0$ is white noise.

Control Allocation Problem

The control allocation problem is to find a mapping from the desired control force, τ_d, to the desired actuator commands, u_d, such that the desired force is realized. Thus, for a given control actuation matrix, $\hat{B}$, the control allocation problem reduces to:

$$\begin{aligned} u_d &:= A\tau_d \\ \text{s.t. } \hat{B}u_d &= \tau_d \end{aligned} \implies \hat{B}A = \mathbb{I}, \tag{6.6}$$

that is, to find a matrix, A, that is a right inverse of $\hat{B}$.

Note that when $\hat{B}A$ is not diagonal, the system will have some control-induced cross coupling between the control axes. If $\hat{B}A$ is diagonal but not the identity matrix, then there will not be any control-induced cross coupling, but the system will not achieve the expected performance.

Overview of NF for Feedback Compensation of Learned and Adapted Dynamics

By following prior work [17], [24], it is straightforward to extend the nominal control law with a learned residual model of the faulty dynamics, $\hat{g}_{\mathrm{NF}}(x, u, t) \approx g(x, u, t)$, that is simultaneously learning the original residual and the actuator failures. However, as we will discuss in this section, this framework can break down in the case of actuator failures. [24] showed that spectral normalization of a learned residual model for g guarantees the existence of a stabilizing control solution and robustness of the augmented control. [17] extended the learned-residual control framework to online learning of dynamics, allowing effective and robust adaptation of a pre-trained model to a time-varying conditions. The learned model of the dynamics is incorporated into the control scheme with the following iteratively updated control law:

$$u_{\mathrm{NF},k} = {B_0}_R^{-1}(-K(x - x_d) + \dot{x}_d - f(x) - \hat{g}_{\mathrm{NF}}(x, u_{k-1}, t)). \tag{6.7}$$

where the fixed point iteration $\hat{g}(x, u_{k-1}, t) \approx \hat{g}(x, u_k, t)$ is used to handle the non-affine control problem that arises from the learned model. To ensure stability, the fixed point iteration requires that the Lipschitz constant of the iteration is less than one, which is true when

$$\sigma({B_0}_R^{-1}) \cdot \mathcal{L}_u(\hat{g}(x, u, t)) < 1 \tag{6.8}$$

where $\sigma({B_0}_R^{-1})$ is the spectral norm of ${B_0}_R^{-1}$ and $\mathcal{L}_u(\hat{g}(x, u, t))$ is the Lipschitz constant of $\hat{g}(x, u, t)$ with respect to u. Furthermore, [24] showed that the exponential convergence rate of the closed-loop system depends on the convergence rate of this fixed-point iteration. In particular, the exponential convergence rate, α, is proportional to $(\lambda_{\max}(K) - \rho)$, where ρ bounds the one-step difference in the control input, such that $\|u_k - u_{k-1}\| \leq \rho\|\tilde{x}_k\| \approx \rho\|\tilde{x}_{k-1}\|$. Because of this Lipschitz constant requirement, the control law in (6.7) can break down in the presence of faults.

6.3 Methods

Control allocation through online optimization

First, we introduce two common choices for control allocation matrix, one prior approach, and then propose a novel control allocation algorithm that maximizes control authority. Because these methods do not explicitly consider the time-varying nature of the system, we will simply denote the control actuation matrix of interest as B.

Moore-Penrose Pseudoinverse Allocation

A natural choice for the control allocation matrix, A, is the Moore-Penrose right pseudoinverse, given by

$$A = B^{\dagger} = B^{\top}\left(BB^{\top}\right)^{-1} \tag{6.9}$$

Note that for controllable overactuated systems, i.e., the type of system we are considering, B is a wide, full row rank matrix, so $(BB^{\top})^{-1}$ is well-defined. This choice of control allocation matrix yields the minimum norm control input given any desired torque command, that is,

$$A_{\text{pinv}} = \begin{array}{cc} \operatorname{argmin}_A \max_{\|\tau\|_2=1} & \|A\tau\|_2 \\ \text{s.t.} & BA = \mathbb{I} \end{array}. \tag{6.10}$$

However, the minimum norm solution does not account for actual power usage or control saturation. Thus, it is not often the best choice.

Maximum Control Authority Allocation

For a symmetric multirotor, we can design a control allocation matrix that maximizes control authority by choosing thrust and torque factors that independently create the maximum thrust and moments. A multirotor is symmetric when $B\text{sign}(B^{\top})$ is diagonal. The maximum torque along the i'th axis is produced when $u_{\tau_{\max,i}} = \max\left(\text{sign}(B^{\top})_{(\cdot,i)}, 0\right)$, where $(B^{\top})_{(\cdot,i)}$ is the i'th column of $B^{\top}$ and max is the element-wise maximum, here. Thus, the allocation matrix that yields maximum control authority along each control axis, independently, is

$$A_{\text{mca}} = \text{sign}(B^\top) \tag{6.11}$$

On most multirotors and every a symmetric multirotor, this allocation scheme will not work under a single motor failure. For example, consider a perfectly sensed motor failure, such that $B = B_0 H(t)$, where B_0 represents the nominal, symmetric system. Any single motor failure will cause $(B_0 H)\text{sign}((B_0 H)^\top)$ to become non-diagonal. This leads to cross coupling in the different control axis and significantly degraded tracking performance. Thus, for fault-tolerant control, we must consider more sophisticated allocation algorithms.

Kim's Control Allocation

[5] proposes the following allocation algorithm:

$$A_{\text{kim}} = \begin{array}{ll} \text{argmin}_A & \|A\|_F + \frac{1}{m}\sum_{i=1}^m |A_{(0,i)} - \text{mean}(A_{(0,\cdot)})| \\ \text{s.t.} & BA = \mathbb{I}, \quad A_{(0,\cdot)} \geq 0 \end{array} \tag{6.12}$$

The first term in the cost function, $\|A\|_F$, is the Frobenius norm of A, which is used as a surrogate for the control effort. The second term distributes the thrust among the motors as evenly as possible. The constraints ensure that the solution is a valid control allocation matrix for B and that the thrust factors are non-negative. However, we find that under an outboard motor failure for our system in Sec. 6.4, some thrust factors are 0 with non-zero torque factors. Thus, there are infinitesimally small torque commands can cause the control to saturate.

Proposed Allocation Algorithm

Do to the limitations of prior approaches, we propose the following allocation algorithm. This method directly maximizes the control authority at a nominal operating point, where the thrust equals the (scaled) weight of the vehicle, m. Furthermore, this formulation is not only convex, but also it is a linear program. Thus, it can be solved efficiently using, for example using the [25], [26]

The thrust for a given set of motor speeds is given by $B_{0(1,\cdot)}u$. Thus, to achieve the maximum thrust with no torque, that is $\tau_d = [1; 0; 0; 0]$, we must have $u_d = A\tau_d = A_{(\cdot,1)}$. Similarly, to achieve the maximum torque along the ith axis while producing m thrust, we must have $u_d = A\tau_d = mA_{(\cdot,1)} + A_{(\cdot,i)}$. Since the vehicle is asymmetric, we must consider both the positive and negative torque along each axis. Accounting for actuation limits, this leads

$$
\begin{aligned}
\bar{A}_{\text{NFF}} = \underset{A}{\text{argmax}} \quad & B_{(1,\cdot)}A_{(\cdot,1)} + \textstyle\sum_{i=2}^{n} B_{(i,\cdot)}\left(mA_{(\cdot,1)} + A_{(\cdot,i)}\right) \\
& - \textstyle\sum_{i=2}^{n} B_{(i,\cdot)}\left(mA_{(\cdot,1)} - A_{(\cdot,i)}\right) \\
\text{s.t.} \quad & B_{(2:4,\cdot)}A_{(\cdot,0)} = 0, A_{(\cdot,1)} \geq 0, A_{(\cdot,1)} \leq 1, \\
& B_{((1,3,4),\cdot)}A_{(\cdot,2)} = [m,0,0]^\top, \\
& 0 \leq \left(mA_{(\cdot,1)} + A_{(\cdot,i)}\right) \leq \mathbb{1}, 0 \leq \left(mA_{(\cdot,1)} - A_{(\cdot,i)}\right) \leq \mathbb{1}, \\
& B_{((1,2,4),\cdot)}A_{(\cdot,3)} = [m,0,0]^\top, \\
& 0 \leq \left(mA_{(\cdot,1)} + A_{(\cdot,i)}\right) \leq \mathbb{1}, 0 \leq \left(mA_{(\cdot,1)} - A_{(\cdot,i)}\right) \leq \mathbb{1}, \\
& B_{((1,2,3),\cdot)}A_{(\cdot,4)} = [m,0,0]^\top, \\
& 0 \leq \left(mA_{(\cdot,1)} + A_{(\cdot,i)}\right) \leq \mathbb{1}, 0 \leq \left(mA_{(\cdot,1)} - A_{(\cdot,i)}\right) \leq \mathbb{1}
\end{aligned}
\tag{6.13}
$$

For failure scenarios, $B\bar{A}_{\text{NFF}} \neq \mathbb{I}$ due to the reduced control authority, however, $B\bar{A}_{\text{NFF}}$ is diagonal. Thus, we simply must rescale $\bar{A}_{\text{NFF}}$ to ensure that $B\bar{A}_{\text{NFF}} = \mathbb{I}$. Thus, the final control allocation matrix is given by

$$A_{\text{NFF}} = \bar{A}_{\text{NFF}}\left(B\bar{A}_{\text{NFF}}\right)^{-1} \tag{6.14}$$

Under nominal conditions, this exactly reproduces the solution from (6.11). Furthermore, under a single motor failure, this algorithm will maintain maximum control authority while maintaining the nominal performance characteristics of the system.

Motor Efficiency Adaptation as an Extension of Learned Dynamics

Consider the following learning architectures and control laws, which are all models for the error in the nominal model, (6.5).

$$\hat{g}_{\text{NF}}(x,u) = \phi(x,u)\hat{a}, \qquad u_{\text{NF}} = (B_0)_R^{-1}(-f(x) - K\tilde{x} - \hat{g}_{\text{NF}}) \tag{6.15}$$

$$\hat{g}_{\text{B}}(x,u) = (\hat{B} - B_0)u, \qquad u_{\text{B}} = \hat{B}_R^{-1}(-f(x) - K\tilde{x}) \tag{6.16}$$

$$\hat{g}_{\text{eff}}(x,u) = B_0(\hat{H} - \mathbb{I})u, \qquad u_{\text{eff}} = (B_0\hat{H})_R^{-1}(-f(x) - K\tilde{x}) \tag{6.17}$$

$$
\begin{aligned}
\hat{g}_{\text{NFF}}(x,u) &= B_0(\hat{H} - \mathbb{I})u + \phi(x,u)\hat{a}, \\
u_{\text{NFF}} &= (B_0\hat{H})_R^{-1}(-f(x) - K\tilde{x} - \phi(x,u)\hat{a})
\end{aligned}
\tag{6.18}
$$

$\hat{g}_{\text{NF}}$ is the learned dynamics model from [17], $\hat{g}_{\text{B}}$ is full actuation matrix adaptation, $\hat{g}_{\text{eff}}$ is motor efficiency adaptation, and $\hat{g}_{\text{NFF}}$ is the proposed method, which combines motor efficiency adaptation and learned dynamics. In the next section, Sec. 6.3, we will discuss online adaptation of the full control actuation matrix, (6.16) and some challenges of this approach. Then, we will continue our analysis only for (6.18), since (6.17) and (6.15) are special cases of (6.18).

Full Actuation Matrix Adaptation

To simplify the notation, define $\bar{B} = \hat{B} - B_0$. Consider the continuous time cost function

$$
\begin{aligned}
J(\bar{B}) &= \int_0^t \mathrm{e}^{-(t-r)/\lambda_1} \left\|y - \bar{B}u\right\|^2 \mathrm{d}r + \lambda_2 \|\bar{B}\|_F^2 && (6.19)\\
&= \int_0^t \mathrm{e}^{-(t-r)/\lambda_1} \mathrm{tr}\left[(y - \bar{B}u)(y - \bar{B}u)^\top\right] \mathrm{d}r + \lambda_2 \mathrm{tr}\left(\bar{B}\bar{B}^\top\right) && (6.20)\\
&= \int_0^t \mathrm{e}^{-(t-r)/\lambda_1} \mathrm{tr}\left[yy^\top - 2\bar{B}uy^\top + \bar{B}uu^\top\bar{B}\right] \mathrm{d}r + \lambda_2 \mathrm{tr}\left(\bar{B}\bar{B}^\top\right) && (6.21)
\end{aligned}
$$

Since this is convex and quadratic in B, we can easily find the solution by looking for the critical point. We are using the following notation: $\left[\frac{\partial J}{\partial \bar{B}}\right]_{ij} = \frac{\partial J}{\partial \bar{B}_{ij}}$.

$$
\begin{aligned}
\frac{\partial J}{\partial \bar{B}} &= \int_0^t \mathrm{e}^{-(t-r)/\lambda_1} \left[0 - 2yu^\top + 2\bar{B}uu^\top\right] \mathrm{d}r + \lambda_2 2\bar{B} && (6.22)\\
\rightsquigarrow \bar{B}\left(\lambda_2 \mathbb{I} + \int_0^t \mathrm{e}^{-(t-r)/\lambda_1} uu^\top \mathrm{d}r\right) &= \int_0^t \mathrm{e}^{-(t-r)/\lambda_1} yu^\top \mathrm{d}r && (6.23)\\
\bar{B} &= \int_0^t \mathrm{e}^{-(t-r)/\lambda_1} yu^\top \mathrm{d}r \underbrace{\left(\lambda_2 \mathbb{I} + \int_0^t \mathrm{e}^{-(t-r)/\lambda_1} uu^\top \mathrm{d}r\right)^{-1}}_{P} && (6.24)
\end{aligned}
$$

Now we can derive a recursive update law for $\bar{B}$. Starting with P,

$$
\begin{aligned}
\dot{P} &= -P\frac{\mathrm{d}\left(P^{-1}\right)}{\mathrm{d}t}P && (6.25)\\
&= -P\left(uu^\top - \frac{1}{\lambda_1}\int_0^t \mathrm{e}^{-(t-r)/\lambda_1} uu^\top \mathrm{d}r\right)P && (6.26)\\
&= -P\left(uu^\top - \frac{1}{\lambda_1}\left(P^{-1} - \lambda_2 \mathbb{I}\right)\right)P && (6.27)\\
\dot{P} &= \frac{1}{\lambda_1}P - P\left(\frac{\lambda_2}{\lambda_1}\mathbb{I} + uu^\top\right)P. && (6.28)
\end{aligned}
$$

Then we can compute $\dot{\bar{B}}$.

$$
\dot{\bar{B}} = \left(yu^\top - \frac{1}{\lambda_1}\int_0^t \mathrm{e}^{-(t-r)/\lambda_1} yu^\top \mathrm{d}r\right)P
$$

$$+ \int_0^t \mathrm{e}^{-(t-r)/\lambda_1} y u^\top \mathrm{d}r \left(\frac{1}{\lambda_1} P - P \left(\frac{\lambda_2}{\lambda_1} \mathbb{I} + u u^\top \right) P \right) \tag{6.29}$$

$$= y u^\top P - \bar{B} \left(u u^\top + \frac{\lambda_2}{\lambda_1} \mathbb{I} \right) P \tag{6.30}$$

$$\dot{\bar{B}} = - \left(\bar{B} u - y \right) u^\top P - \frac{\lambda_2}{\lambda_1} \bar{B} P \tag{6.31}$$

As we will see later, it is useful to consider a composite adaptation law, that is an adaptation law that depends on both $\bar{B}u - y$ and s, given by

$$\dot{\bar{B}} = - \left(\bar{B} u - y \right) u^\top P - \frac{\lambda_2}{\lambda_1} \bar{B} P + s u^\top P \tag{6.32}$$

The closed loop dynamics are given by

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = Bu \tag{6.33}$$

$$u = \bar{B}^\dagger (M(q)\ddot{q}_r + C(q, \dot{q})\dot{q}_r + g(q) - Ks) \tag{6.34}$$

$$\tilde{B} = \bar{B} + B_0 - B \leftrightarrow B = \bar{B} + B_0 - \tilde{B} \tag{6.35}$$

$$M\dot{s} + (C + K)s = -\tilde{B}(\bar{B} + B_0)(M\ddot{q}_r + C\dot{q} + g(q) - Ks) \tag{6.36}$$

Take the following Lyapunov function

$$\mathcal{V} = s^\top M(q) s + \|\tilde{B}\|^2_{\mathrm{F},P^{-1}} \tag{6.37}$$

$$\|\tilde{B}\|^2_{\mathrm{F},P^{-1}} \triangleq \operatorname{tr} \left[\tilde{B} P^{-1} \tilde{B}^\top \right] \tag{6.38}$$

$$\mathcal{V}(s, \tilde{B}) = s^\top M(q) s + \operatorname{tr} \left[\tilde{B} P^{-1} \tilde{B}^\top \right] \tag{6.39}$$

then

$$\dot{\mathcal{V}} = 2 s^\top M \dot{s} + s \top \dot{M} s + 2\operatorname{tr} \left(\tilde{B} P^{-1} \dot{\bar{B}}^\top \right) + \operatorname{tr} \left(\tilde{B} \frac{\mathrm{d}}{\mathrm{d}t} \left(P^{-1} \right) \tilde{B}^\top \right) \tag{6.40}$$

$$\begin{aligned} &= 2 s^\top \left(-(C + K)s - \tilde{B} u \right) + s^\top M s \\ &\quad + 2\operatorname{tr} \left(\tilde{B} P^{-1} \left(- \left(\tilde{B} u \right) u^\top P - \frac{\lambda_2}{\lambda_1} \bar{B} P + s u^\top P \right)^\top \right) \\ &\quad + \operatorname{tr} \left(\tilde{B} \left(u u^\top - \frac{1}{\lambda_1} \left(P^{-1} - \lambda_2 \mathbb{I} \right) \right) \tilde{B}^\top \right) \end{aligned} \tag{6.41}$$

$$\begin{aligned} &= -2 s^\top K s - 2 s^\top \tilde{B} u + 2\operatorname{tr} \left(\tilde{B} P^{-1} \left(s u^\top P \right)^\top \right) \\ &\quad + 2\operatorname{tr} \left(\tilde{B} \left(-u u^\top \tilde{B}^\top - \frac{\lambda_2}{\lambda_1} \bar{B}^\top \right) \right) \end{aligned} \tag{6.42}$$

$$
\begin{aligned}
&+2\operatorname{tr}\left(\frac{\lambda_2}{\lambda_1}\tilde{B}(B-B_0)^\top\right)-2\operatorname{tr}\left(\frac{\lambda_2}{\lambda_1}\tilde{B}(B-B_0)^\top\right)\\
&+\operatorname{tr}\left(\tilde{B}uu^\top\tilde{B}^\top-\frac{1}{\lambda_1}\tilde{B}P^{-1}\tilde{B}^\top+\frac{\lambda_2}{\lambda_1}\tilde{B}\tilde{B}^\top\right)\\
&=-2s^\top Ks-2\operatorname{tr}\left(\frac{\lambda_2}{\lambda_1}\tilde{B}(B-B_0)^\top\right) \qquad (6.43)\\
&+\operatorname{tr}\left(-\tilde{B}uu^\top\tilde{B}^\top-\frac{1}{\lambda_1}\tilde{B}P^{-1}\tilde{B}^\top-\frac{\lambda_2}{\lambda_1}\tilde{B}\tilde{B}^\top\right)\\
\dot{\mathcal{V}}&=-2s^\top Ks-\operatorname{tr}\left(\tilde{B}\left(uu^\top+\frac{1}{\lambda_1}P^{-1}+\frac{\lambda_2}{\lambda_1}\mathbb{I}\right)\tilde{B}^\top\right) \qquad (6.44)\\
&-2\operatorname{tr}\left(\frac{\lambda_2}{\lambda_1}\tilde{B}(B-B_0)^\top\right)
\end{aligned}
$$

Lemma 6.3.1. *Note that for matrices $B\in\mathbb{R}^{n\times m}$, $C\in\mathbb{R}^{m\times m}$, and $D\in\mathbb{R}^{m\times m}$, if $D>C$ and rank$(B)=n$, then* $\operatorname{tr}\left(B(D-C)B^\top\right)>0$.

Proof. For any $x\in\mathbb{R}^n$, if $x\neq 0$ and $\operatorname{rank}(B)=n$ then $B^\top x\neq 0$. When $D-C>0$, we also have the $x^\top B(D-C)B^\top x>0$, and thus $B(D-C)B^\top>0$. Since the trace of a matrix is equal to the sum of the eigenvalues of a matrix, and all the eigenvalues of a positive definite matrix are positive, $\operatorname{tr}\left(B(D-C)B^\top\right)>0$. □

Define $\alpha>0$ as the exponential convergence rate of the system such that

$$\left(uu^\top+\frac{1}{\lambda_1}P^{-1}+\frac{\lambda_2}{\lambda_1}\mathbb{I}\right)>2\alpha P^{-1}\quad\text{and} \qquad (6.45)$$

$$K>\alpha M. \qquad (6.46)$$

Aside. *We can slightly tighten the convergence bound since $D-C>0$ is sufficient but not necessary for* $\operatorname{tr}\left(B(D-C)B^\top\right)>0$. *In particular, (6.45) can be loosened to*

$$\sum_i \operatorname{eig}_i\left(\left(uu^\top+\frac{1}{\lambda_1}P^{-1}+\frac{\lambda_2}{\lambda_1}\mathbb{I}\right)-2\alpha P^{-1}\right)>0 \qquad (6.47)$$

where eig_i is the i'th eigenvalue of the matrix.

Define $D=\frac{\lambda_2}{\lambda_1}\left\|P^{1/2}(B-B_0)^\top\right\|$. Then

$$\dot{\mathcal{V}}\leq-2\alpha\mathcal{V}+2\sqrt{\mathcal{V}}D \qquad (6.48)$$

Consider the related system $\mathcal{W}$ where $\mathcal{W}=\sqrt{\mathcal{V}}$ and $2\mathcal{W}\dot{\mathcal{W}}=\dot{V}$. Then, from (6.48),

$$2\mathcal{W}\dot{\mathcal{W}}\leq-2\alpha\mathcal{W}^2+2\mathcal{W}D \qquad (6.49)$$

$$\dot{\mathcal{W}} \leq -\alpha \mathcal{W} + D. \tag{6.50}$$

Consider another related system, $w(t)$, defined by $\dot{w}(t) = -\alpha w(t) + D(t)$ and $w(0) = \mathcal{W}(0)$. The solution to $w(t)$ is

$$w(t) = \mathrm{e}^{-\alpha t} w(0) + \int_0^t \mathrm{e}^{-\alpha(t-r)} D(r) \mathrm{d}r, \tag{6.51}$$

which can be bounded by

$$w(t) \leq \mathrm{e}^{-\alpha t} \left(w(0) - \sup_t \frac{D(t)}{\alpha} \right) + \sup_t \frac{D(t)}{\alpha} \tag{6.52}$$

By the Comparison Lemma [27],

$$\sqrt{\mathcal{V}} = \mathcal{W} \leq w(t), \tag{6.53}$$

thus $\sqrt{\mathcal{V}}$ and also $\|\tilde{x}\|$ exponentially converges to the ball

$$\|\tilde{x}\| \leq \sqrt{\mathcal{V}} \leq sup_t \frac{D}{\alpha} \tag{6.54}$$

While this shows stability of the system, convergence of the system can be slow. This is an inherent limitation of directly adapting all parameters of the control actuation matrix. Furthermore, this method can be sensitive to noise or non-zero f_{res}, which we have not considered here. In the next section, we will focus on adaptation of the efficiency factors, which enables faster adaptation, and therefore faster convergence.

Kalman Filter Based Adaptation and $\ell 2$ Regularized Least Squares

With some simple rearrangements, we can write the Kalman Filter based composite adaptation law following [17]. To see that the Kalman filter adaptation will follow [17], consider the following rearrangements.

$$\hat{g}_{\text{NFF}} = \begin{bmatrix} B_0 U & \phi \end{bmatrix} \begin{bmatrix} \hat{\eta} - \mathbb{1} \\ \hat{a} \end{bmatrix} \tag{6.55}$$

$$\text{where} \quad \hat{\eta} = \operatorname{diag}(\hat{H}), \text{ and} \tag{6.56}$$

$$U = \operatorname{diag}(u) \tag{6.57}$$

Then, the Kalman filter based adaptation law is given by

$$\begin{bmatrix} \dot{\hat{\eta}} \\ \dot{\hat{a}} \end{bmatrix} = -\omega_f \begin{bmatrix} \hat{\eta} - \mathbb{1} \\ \hat{a} \end{bmatrix} + P \begin{bmatrix} B_0 U & \phi \end{bmatrix}^\top R^{-1} \left(y - \begin{bmatrix} B_0 U & \phi \end{bmatrix} \begin{bmatrix} \hat{\eta} - \mathbb{1} \\ \hat{a} \end{bmatrix} \right)$$

$$+ P \begin{bmatrix} B_0 U & \phi \end{bmatrix}^\top \tilde{x} \tag{6.58}$$

$$\dot{P} = -2\omega_f P + Q - P \left(\begin{bmatrix} B_0 U & \phi \end{bmatrix}^\top R^{-1} \begin{bmatrix} B_0 U & \phi \end{bmatrix} \right) P \tag{6.59}$$

A similar $\ell 2$-regularized least squares with exponential forgetting formulation can also be derived, which takes the form

$$\begin{bmatrix} \dot{\hat{\eta}} \\ \dot{\hat{a}} \end{bmatrix} = -\gamma \begin{bmatrix} \hat{\eta} - \mathbb{1} \\ \hat{a} \end{bmatrix} + P \begin{bmatrix} B_0 U & \phi \end{bmatrix}^\top R^{-1} \left(y - \begin{bmatrix} B_0 U & \phi \end{bmatrix} \begin{bmatrix} \hat{\eta} - \mathbb{1} \\ \hat{a} \end{bmatrix} \right)$$

$$+ P \begin{bmatrix} B_0 U & \phi \end{bmatrix}^\top \tilde{x} \tag{6.60}$$

$$\dot{P} = \omega_f P - P \left(\begin{bmatrix} B_0 U & \phi \end{bmatrix}^\top \begin{bmatrix} B_0 U & \phi \end{bmatrix} + \Gamma \right) P \tag{6.61}$$

where Γ is a diagonal positive definite matrix that controls the regularization cost in the least squares problem. Note that the closed from solution for P is given by

$$P \equiv \left(\int_0^t e^{-\omega_f (t-r)} \begin{bmatrix} B_0 U & \phi \end{bmatrix}^\top \begin{bmatrix} B_0 U & \phi \end{bmatrix} dr + \Gamma \right)^{-1}. \tag{6.62}$$

The proof of stability largely follows that of [17] once the closed loop dynamics have been sufficiently rearranged, as we do below in (6.85). There are two added complexities in the proof, which are that the disturbance term becomes a function of $U^\top d$ and that uniform boundedness of P now depends on uniform boundedness of U. Although we will omit the proof, and address these challenges for the $\ell 1$-regularized adaptation law in the next section; the proof for the $\ell 2$-regularized adaptation law and Kalman filter adaptation law follows exactly the same form, except for the form of the regularization term and P update equation.

The $\ell 2$-regularized and Kalman-filter-based methods are not necessarily able to correctly identify the underlying faults, but the estimated efficiencies vector is sufficient to stabilize and re-balance the system. Both of these results are a result of a lack of persistent excitation. Because we are considering an over-actuated system, and because we control the design of the control allocation matrix, the control allocation can be perturbed to obtain persistent excitation without affecting tracking performance. This would require constantly updating the allocation scheme to excite different modes in the system, while still satisfying the key allocation constraint, (6.6).

Instead, we will consider an alternate regularization method in the following section, which encourages sparse failure identification.

Sparse Failure Identification

In this section, we will consider an update policy similar to the $\ell 2$-regularized adaptive update law in the last section, except we will use an ℓ_1-regularized update policy. This is a common regularization term for sparse parameter estimation, because it encourages sparse solutions without requiring a hard constraint on the number of non-zero parameters or iteration through many non-zero parameter combinations.

Discrete Update Law

Consider the following least squares loss function.

$$J_k(\hat{\eta}, \hat{a}) = \sum_{i=0}^{k} \mathrm{e}^{(-\omega_f(t_k - t_i))} \left\|y_i - \hat{g}_{\mathrm{NFF}}\right\|_2^2 + \gamma_\eta \left\|\hat{\eta} - 1\right\|_1 + \gamma_a \left\|\hat{a}\right\|_2^2 \tag{6.63}$$

First, simplify the loss function by moving $\hat{\eta}$ and $\hat{a}$ outside the summation and defining $\bar{\eta} = \hat{\eta} - 1$.

$$\begin{aligned}
J_k(\hat{\eta}, \hat{a}) &= \sum_{i=0}^{k} \mathrm{e}^{(-\omega_f(t_k - t_i))} (y_i - \hat{g})^\top (y_i - \hat{g}) + \gamma_\eta \left\|\bar{\eta}\right\|_1 + \gamma_a \left\|\hat{a}\right\|_2^2 && (6.64)\\
&= \sum_{i=0}^{k} \mathrm{e}^{(-\omega_f(t_k - t_i))} \left(y_i^\top y_i - 2 y_i^\top \hat{g} + \hat{g}^\top \hat{g}\right) + \gamma_\eta \left\|\bar{\eta}\right\|_1 + \gamma_a \left\|\hat{a}\right\|_2^2 && (6.65)\\
&= \left(\sum_{i=0}^{k} \mathrm{e}^{(-\omega_f(t_k - t_i))} y_i^\top y_i\right) \\
&\quad - 2\left(\sum_{i=0}^{k} \mathrm{e}^{(-\omega_f(t_k - t_i))} y_i^\top \begin{bmatrix} B_0 U_i & \phi_i \end{bmatrix}\right) \begin{bmatrix} \bar{\eta} \\ \hat{a} \end{bmatrix} \\
&\quad + \begin{bmatrix} \bar{\eta}^\top & \hat{a}^\top \end{bmatrix} \left(\sum_{i=0}^{k} \mathrm{e}^{(-\omega_f(t_k - t_i))} \begin{bmatrix} U_i B_0^\top B_0 U_i & U_i B_0^\top \phi_i \\ \phi_i^\top B_0 U_i & \phi_i^\top \phi_i \end{bmatrix}\right) \begin{bmatrix} \bar{\eta} \\ \hat{a} \end{bmatrix} \\
&\quad + \gamma_\eta \left\|\bar{\eta}\right\|_1 + \gamma_a \left\|\hat{a}\right\|_2^2 && (6.66)
\end{aligned}$$

This is a convex function of $\bar{\eta}$ and $\hat{a}$, so we can easily solve for the optimal $\bar{\eta}$ and $\hat{a}$ using a number of numerical solving tools. During online computation, we can quickly incorporate a new measurement by scaling the old summation terms by $\exp\left(-\omega_f(t_k - t_{k-1})\right)$ and adding the k'th term. Then we can solve for the optimal $\bar{\eta}$ and $\hat{a}$ using a numerical solver as needed. Note, that we can also easily derive a recursive solution for the optimal $\hat{a}$, with a simplified update step to quickly incorporate new measurements and a step for computation of the optimal $\hat{a}$ given the most recently computed $\bar{\eta}$, which would require solving a linear system of equations.

Continuous Update Law

The analogous continuous time adaptation law for $\hat{\eta}$ can be found from the following cost function:

$$J(\bar{\eta}) = \int_0^t \mathrm{e}^{-\omega_f(t-r)} \, \|y - BU(r)\bar{\eta}\|^2 \, \mathrm{d}r + 2\gamma\|\bar{\eta}\|_1 \tag{6.67}$$

$$= \int_0^t \mathrm{e}^{-\omega_f(t-r)} \left(y^\top y - 2y^\top BU\bar{\eta} + \bar{\eta}^\top UB^\top BU\bar{\eta}\right) \mathrm{d}r + 2\gamma\|\bar{\eta}\|_1 \tag{6.68}$$

Note that we have dropped back to the simpler case of (6.17), though the full continuous time stability analysis for (6.18) follows similarly under the assumption of Lipschitz boundedness of ϕ.

First, approximate the ℓ_1 norm such that

$$\|\bar{\eta}\|_1 \approx \sum_i \sqrt{\bar{\eta}_i^2 + \epsilon} = \|\bar{\eta}\|_{1,\epsilon} \text{ and } \lim_{\epsilon\to 0} \sum_i \sqrt{\bar{\eta}_i^2 + \epsilon} = \|\bar{\eta}\|_1 \tag{6.69}$$

Then the cost function in (6.68) is approximated $J(\bar{\eta}, \epsilon)$ such that $\lim_{\epsilon\to 0} J(\bar{\eta}, \epsilon) = J(\bar{\eta})$, where $J(\bar{\eta}, \epsilon)$ is given by

$$\begin{aligned} J(\bar{\eta}) \approx J(\bar{\eta}, \epsilon) = \int_0^t \mathrm{e}^{-\omega_f(t-r)} \left(y^\top y - 2y^\top B_0U\bar{\eta} + \bar{\eta}^\top UB_0^\top B_0U\bar{\eta}\right) \mathrm{d}r \\ + 2\gamma \sum_i \sqrt{\bar{\eta}_i^2 + \epsilon} \end{aligned} \tag{6.70}$$

Since this cost function is convex in $\bar{\eta}$, the minimum value is obtained when $\frac{\partial J}{\partial \bar{\eta}} = 0$, as follows.

$$\frac{\partial J}{\partial \bar{\eta}} = \int_0^t \mathrm{e}^{-\omega_f(t-r)} \left(-2UB_0^\top y + 2UB_0^\top B_0U\bar{\eta}\right) \mathrm{d}r + 2\gamma \left[\frac{\bar{\eta}_1}{\sqrt{\bar{\eta}_1^2 + \epsilon}}, \frac{\bar{\eta}_2}{\sqrt{\bar{\eta}_2^2 + \epsilon}}, \cdots\right] \tag{6.71}$$

$$\frac{\partial J}{\partial \bar{\eta}} = 0 \rightsquigarrow$$

$$\left(\gamma \mathrm{diag}\left(\left[\frac{1}{\sqrt{\bar{\eta}_1^2 + \epsilon}}, \frac{1}{\sqrt{\bar{\eta}_2^2 + \epsilon}}, \cdots\right]\right) + \int_0^t \mathrm{e}^{-\omega_f(t-r)} U(r)B_0^\top B_0U(r)\mathrm{d}r\right)\bar{\eta} = \int_0^t \mathrm{e}^{-\omega_f(t-r)} U(r)B_0^\top y\mathrm{d}r \tag{6.72}$$

$$\bar{\eta} = \underbrace{\left(\gamma \mathrm{diag}\left(\left[\frac{1}{\sqrt{\bar{\eta}_1^2+\epsilon}}, \frac{1}{\sqrt{\bar{\eta}_2^2+\epsilon}}, \cdots\right]\right) + \int_0^t \mathrm{e}^{-\omega_f(t-r)} U(r) B_0^\top B_0 U(r) \mathrm{d}r\right)^{-1}}_{P}$$
$$\cdot \int_0^t \mathrm{e}^{-\omega_f(t-r)} U(r) B_0^\top y \mathrm{d}r \tag{6.73}$$

Note for some i, if $\bar{\eta}_i = 0$ (typically for ℓ_1 norm minimization), then $\frac{1}{\sqrt{\bar{\eta}_i^2+\epsilon}} = \frac{1}{\sqrt{\epsilon}}$. Otherwise, as $\lim_{\epsilon\to 0} \frac{1}{\sqrt{\bar{\eta}_i^2+\epsilon}} = \frac{1}{|\bar{\eta}_i|}$.

Now we can derive a recursive update law for $\bar{\eta}$. Starting with P:

$$\begin{aligned}
\dot{P} &= -P\frac{\mathrm{d}\left(P^{-1}\right)}{\mathrm{d}t}P \\
&= -P\left(\gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1}{\left(\bar{\eta}_1^2+\epsilon\right)^{3/2}}, \cdots\right]\right) + U(t) B_0^\top B_0 U(t)\right. \\
&\qquad \left. -\omega_f \int_0^t \mathrm{e}^{-\omega_f(t-r)} U(r) B_0^\top B_0 U(r) \mathrm{d}r\right) P \\
&= -P\left(\gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1}{\left(\bar{\eta}_1^2+\epsilon\right)^{3/2}}, \cdots\right]\right) + U(t) B_0^\top B_0 U(t)\right. \\
&\qquad \left. -\omega_f \left(P^{-1} - \gamma \mathrm{diag}\left(\left[\frac{1}{\sqrt{\bar{\eta}_1^2+\epsilon}}, \frac{1}{\sqrt{\bar{\eta}_2^2+\epsilon}}, \cdots\right]\right)\right)\right) P
\end{aligned} \tag{6.74}$$

$$\dot{P} = \omega_f P - P\left(\gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2+\epsilon\right)^{3/2}}, \cdots\right]\right) + U(t)^\top B_0^\top B_0 U(t)\right) P. \tag{6.75}$$

Then we can compute $\dot{\bar{\eta}}$.

$$\begin{aligned}
\dot{\bar{\eta}} &= P\left(U(t)^\top B_0^\top y(t) - \omega_f \int_0^t \mathrm{e}^{-\omega_f(t-r)} U(r) B_0^\top y(r) \mathrm{d}r\right) \\
&\quad + \left(\omega_f P - P\left(\gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2+\epsilon\right)^{3/2}}, \cdots\right]\right) + U(t) B_0^\top B_0 U(t)\right) P\right) \\
&\qquad \cdot \left(\int_0^t \mathrm{e}^{-\omega_f(t-r)} U(r) B_0^\top y(r) \mathrm{d}r\right)
\end{aligned} \tag{6.76}$$

$$= P\left(U(t)^\top B_0^\top y - \omega_f P^{-1}\bar{\eta}\right) \tag{6.77}$$

$$+\left(\omega_f P - P\left(\gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f\bar{\eta}_1^2 + \omega_f\epsilon}{\left(\bar{\eta}_1^2+\epsilon\right)^{3/2}}, \cdots\right]\right) + U(t)B_0^\top B_0 U(t)\right)P\right)P^{-1}\bar{\eta}$$

$$= PU(t)^\top B_0^\top y - \omega_f\bar{\eta} + \omega_f\bar{\eta}$$

$$- P\left(\gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f\bar{\eta}_1^2 + \omega_f\epsilon}{\left(\bar{\eta}_1^2+\epsilon\right)^{3/2}}, \cdots\right]\right) + U(t)B_0^\top B_0 U(t)\right)\bar{\eta} \tag{6.78}$$

$$\dot{\bar{\eta}} = PU(t)^\top B_0^\top (y - B_0 U(t)\bar{\eta}) - \gamma P\cdot \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f\bar{\eta}_1^2 + \omega_f\epsilon}{\left(\bar{\eta}_1^2+\epsilon\right)^{3/2}}, \cdots\right]\right)\cdot\bar{\eta} \tag{6.79}$$

Stability Analysis

Again, for the continuous stability analysis we will focus on (6.17), however, the analysis for (6.18) follows similarly. Assume that we design a control allocation matrix A such that $B_0\hat{H}A = \mathbb{I}$. The closed loop dynamics are

$$\dot{x} = f_n(x) + Bu_{\text{eff}} + g(x,u,t) \tag{6.80}$$

$$= f_n(x) + B_0 Hu + d(t) \tag{6.81}$$

$$= f_n(x) + B_0 H (B_0\hat{H})_R^{-1}(-K(x-x_d) + \dot{x}_d) + d(t) \tag{6.82}$$

$$= f_n(x) + \left(B_0\hat{H}(B_0\hat{H})_R^{-1} + B_0(H-\hat{H})(B_0\hat{H})_R^{-1}\right)(-K(x-x_d) + \dot{x}_d - f_n(x)) \tag{6.83}$$

$$= -K(x-x_d) + \dot{x}_d - B_0\tilde{H}u \tag{6.84}$$

$$\dot{\tilde{x}} = -K\tilde{x} - B_0 U\tilde{\eta} + d(t) \tag{6.85}$$

where $\tilde{\eta} = \hat{\eta} - \eta$, $\tilde{x} = x - x_d$, and $g(x,u,t)$ has been lumped into $d(t)$.

As we will see later, it is useful to consider a composite adaptation law, which is a modification of the adaptation law from (6.79) that includes both $B_0 U\bar{\eta} - y$ and $\tilde{x}$, given by

$$\dot{\bar{\eta}} = PUB_0^\top(y - B_0U\bar{\eta}) - \gamma P\mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f\bar{\eta}_1^2 + \omega_f\epsilon}{\left(\bar{\eta}_1^2+\epsilon\right)^{3/2}}, \cdots\right]\right)\cdot\bar{\eta} + PUB_0^\top\tilde{x} \tag{6.86}$$

Stability can be shown with the following Lyapunov function:

$$\mathcal{V} = \tilde{x}^\top\tilde{x} + \tilde{\eta}^\top P^{-1}\tilde{\eta} = \begin{bmatrix}\tilde{x} & \tilde{\eta}\end{bmatrix}^\top \mathcal{M}\begin{bmatrix}\tilde{x} & \tilde{\eta}\end{bmatrix}, \tag{6.87}$$

$$\text{where} \quad \mathcal{M} = \begin{bmatrix} \mathbb{I} & 0 \\ 0 & P^{-1} \end{bmatrix} \tag{6.88}$$

The derivative is computed as follows:

$$\dot{\mathcal{V}} = 2\tilde{x}^\top \dot{\tilde{x}} + 2\tilde{\eta}^\top P^{-1} \dot{\tilde{\eta}} + \tilde{\eta}^\top \frac{\mathrm{d}\left(P^{-1}\right)}{\mathrm{d}t} \tilde{\eta} \tag{6.89}$$

$$\begin{aligned} &= -2\tilde{x}^\top K \tilde{x} - 2\tilde{x}^\top B_0 U \tilde{\eta} \\ &\quad 2\tilde{\eta}^\top P^{-1} \left(PUB_0^\top (y - B_0 U \bar{\eta}) - \gamma P \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) \cdot \bar{\eta} \right. \\ &\qquad \left. + PUB_0^\top \tilde{x} - \dot{\eta} \right) \\ &\quad + \tilde{\eta}^\top \left(-\omega_f P^{-1} + \gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) + U(t) B_0^\top B_0 U(t) \right) \tilde{\eta} \end{aligned} \tag{6.90}$$

$$\begin{aligned} &= -2\tilde{x}^\top K \tilde{x} - 2\tilde{x}^\top B_0 U \tilde{\eta} + 2\tilde{x}^\top d \\ &\quad + 2\tilde{\eta}^\top U B_0^\top (d - B_0 U \tilde{\eta}) - 2\gamma \tilde{\eta}^\top \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) \cdot \bar{\eta} \\ &\quad + 2\tilde{\eta}^\top U B_0^\top \tilde{x} - 2\tilde{\eta}^\top P^{-1} \dot{\eta} - \omega_f \tilde{\eta}^\top P^{-1} \tilde{\eta} \\ &\quad + \gamma \tilde{\eta}^\top \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) \tilde{\eta} + \tilde{\eta}^\top U B_0^\top B_0 U \tilde{\eta} \end{aligned} \tag{6.91}$$

$$\begin{aligned} &= -2\tilde{x}^\top K \tilde{x} + 2\tilde{x}^\top d \\ &\quad + 2\tilde{\eta}^\top U B_0^\top (d - B_0 U \tilde{\eta}) - 2\gamma \tilde{\eta}^\top \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) \cdot (\tilde{\eta} + \eta) \\ &\quad - 2\tilde{\eta}^\top P^{-1} \dot{\eta} - \omega_f \tilde{\eta}^\top P^{-1} \tilde{\eta} + \gamma \tilde{\eta}^\top \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) \tilde{\eta} \end{aligned} \tag{6.92}$$

$$\begin{aligned} &= -2\tilde{x}^\top K \tilde{x} + 2\tilde{x}^\top d \\ &\quad - \tilde{\eta}^\top \left(U B_0^\top B_0 U - \gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) - \omega_f P^{-1} \right) \tilde{\eta} \end{aligned}$$

$$+ 2\tilde{\eta}^\top \left(U B_0^\top d - \gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) \cdot \eta - P^{-1}\dot{\eta} \right) \tag{6.93}$$

$$= - \begin{bmatrix} \tilde{x} \\ \tilde{\eta} \end{bmatrix}^\top \begin{bmatrix} 2K & 0 \\ 0 & U B_0^\top B_0 U + \gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) + \omega_f P^{-1} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{\eta} \end{bmatrix}$$

$$+ 2 \begin{bmatrix} \tilde{x} \\ \tilde{\eta} \end{bmatrix}^\top \begin{bmatrix} d \\ U B_0^\top d - \gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) \cdot \eta - P^{-1}\dot{\eta} \end{bmatrix} \tag{6.94}$$

Following from the definition of P^{-1}, P^{-1} is bounded and uniformly positive definite. Thus, there exists some $\alpha > 0$ such that

$$- \begin{bmatrix} 2K & 0 \\ 0 & U B_0^\top B_0 U + \gamma \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) + \omega_f P^{-1} \end{bmatrix} \leq -2\alpha \begin{bmatrix} \mathbb{I} & 0 \\ 0 & P^{-1} \end{bmatrix} \tag{6.95}$$

Note that, when P^{-1} is symmetric, uniformly bounded, and uniformly positive definite, $P^{-1/2}$ and $P^{1/2}$ exist and are symmetric, uniformly positive definite, and uniformly bounded. Using (6.94) and (6.95) and the Cauchy-Schwartz inequality, $\dot{\mathcal{V}}$ can be bounded, as follows:

$$\dot{\mathcal{V}} \leq -2\alpha \begin{bmatrix} \tilde{x} \\ \tilde{\eta} \end{bmatrix}^\top \begin{bmatrix} \mathbb{I} & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{\eta} \end{bmatrix}$$
$$+ 2 \left\| \begin{bmatrix} \mathbb{I} & 0 \\ 0 & P^{-1/2} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{\eta} \end{bmatrix} \right\|$$
$$\cdot \left\| \begin{bmatrix} d \\ P^{1/2} U B_0^\top d - \gamma P^{1/2} \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) \cdot \eta - P^{-1/2}\dot{\eta} \end{bmatrix} \right\| \tag{6.96}$$

$$= -2\alpha V + 2\sqrt{V} D \tag{6.97}$$

$$\text{where} \quad D = \left\| \begin{bmatrix} d \\ P^{1/2} U B_0^\top d - \gamma P^{1/2} \mathrm{diag}\left(\left[\frac{\bar{\eta}_1 + \omega_f \bar{\eta}_1^2 + \omega_f \epsilon}{\left(\bar{\eta}_1^2 + \epsilon\right)^{3/2}}, \cdots \right] \right) \cdot \eta - P^{-1/2}\dot{\eta} \end{bmatrix} \right\|$$

Consider the related system $\mathcal{W}$ where $\mathcal{W} = \sqrt{\mathcal{V}}$ and $2\mathcal{W}\dot{\mathcal{W}} = \dot{V}$. Then, from (6.97),

$$2\mathcal{W}\dot{\mathcal{W}} \leq -2\alpha\mathcal{W}^2 + 2\mathcal{W}D \tag{6.98}$$

$$\dot{\mathcal{W}} \leq -\alpha\mathcal{W} + D. \tag{6.99}$$

Consider another related system, $w(t)$, defined by $\dot{w}(t) = -\alpha w(t) + D(t)$ and $w(0) = \mathcal{W}(0)$. The solution to $w(t)$ is

$$w(t) = \mathrm{e}^{-\alpha t}w(0) + \int_0^t \mathrm{e}^{-\alpha(t-r)}D(r)\mathrm{d}r, \tag{6.100}$$

which can be bounded by

$$w(t) \leq \mathrm{e}^{-\alpha t}\left(w(0) - \sup_t \frac{D(t)}{\alpha}\right) + \sup_t \frac{D(t)}{\alpha} \tag{6.101}$$

By the Comparison Lemma [27],

$$\sqrt{\mathcal{V}} = \mathcal{W} \leq w(t), \tag{6.102}$$

thus $\sqrt{\mathcal{V}}$ and also $\|\tilde{x}\|$ exponentially converges to the ball

$$\|\tilde{x}\| \leq \sqrt{\mathcal{V}} \leq sup_t \frac{D}{\alpha} \tag{6.103}$$

Seemingly, the proof is complete at this point. However, we have not yet shown that D is bounded. By assumption, η, d, and $\dot{\eta}$ are uniformly bounded, and B_0, γ, ω_f, and ϵ are constants. D is uniformly bounded if $P^{1/2}$, $P^{-1/2}$, $\bar{\eta}$, and U are initially bounded and continuous.

$P^{1/2}$ and $P^{-1/2}$ are uniformly bounded if P^{-1} is uniformly positive definite and uniformly bounded, respectively. Uniform positive definiteness is guaranteed by uniform positive definiteness of $\bar{\eta}$. Uniform boundedness is guaranteed by uniform boundedness of U and $\bar{\eta}$.

$\bar{\eta}$ is uniformly bounded if η is uniformly bounded and $\tilde{\eta}$ is uniformly bounded.

U is uniformly bounded if $\tilde{x}$ is uniformly bounded and $(B_0\bar{\eta})_R^{-1}$ is uniformly bounded. For the case of (6.18) U also is a function of ϕ, leading to the additional condition that ϕ be bounded, which can be guaranteed if ϕ is Lipschitz bounded.

While precise conditions for uniform boundedness of $(B_0\bar{\eta})_R^{-1}$ is difficult to write out, it is clear that $\tilde{\eta} \to 0$ as $\gamma \to \infty$. We also observe that $P^{1/2} \sim U, \bar{\eta}$, so

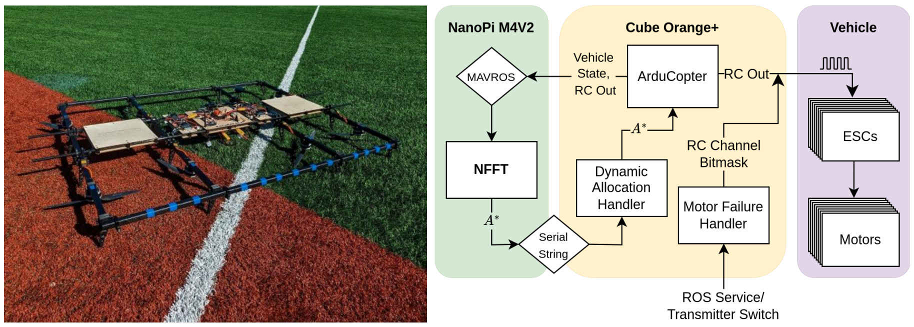


Figure 6.1: **The test aircraft vehicle design** (Left) picture of the vehicle. (Right) schematic of the implemented system. This figure was provided by Joshua Cho.

for small γ, D will be dominated by the term $P^{1/2}UB_0^\top d$. For sufficiently large γ, $(B_0\bar{\eta})_R^{-1}$ is bounded. Lastly, for very large γ, no adaptation will occur, and the system will maintain the baseline performance. Thus, there is an inherent design trade off between the degree of regularization and the nominal modeling errors not captured by the efficiency adaptation model, in the case of (6.17), or the learning representation error in the case of (6.18).

6.4 Experimental Validation

Hardware Setup

Neural-Fly for Fault Tolerancewas deployed on a custom-built octocopter (Fig. 6.1, left) based upon the Autonomous Flying Ambulance [28], an eVTOL UAV specifically designed to maximize controllability when one or more motors fail. The octocopter has a maximum motor-to-motor distance of 1.57 m and has a mass of approximately 8.58 kg. The aircraft is stabilized using ArduCopter [29] running on a Cube Orange+, and uses IMUs, barometers, magnetometers and an RTK GNSS to provide the state estimate. The motors are controlled via Dshot with telemetry, allowing the RPM of the motors to be measured directly from the ESCs, providing near-instant feedback of the achieved motor speed. The aircraft is configured as a Dynamic Scripting Matrix frame class, allowing the control allocation matrix to be updated in-flight.

A NanoPi M4V2 is used as the companion computer, allowing NFF to run entirely onboard the aircraft. The ROS2-enabled companion computer receives state estimate information and mirrored PWM commands from ArduCopter via standard interfacing. The transfer of the control allocation matrix to ArduCopter is implemented through a dedicated UART port between the two computers, with a LUA script on the Cube Orange+ capturing, verifying, and updating control allocation

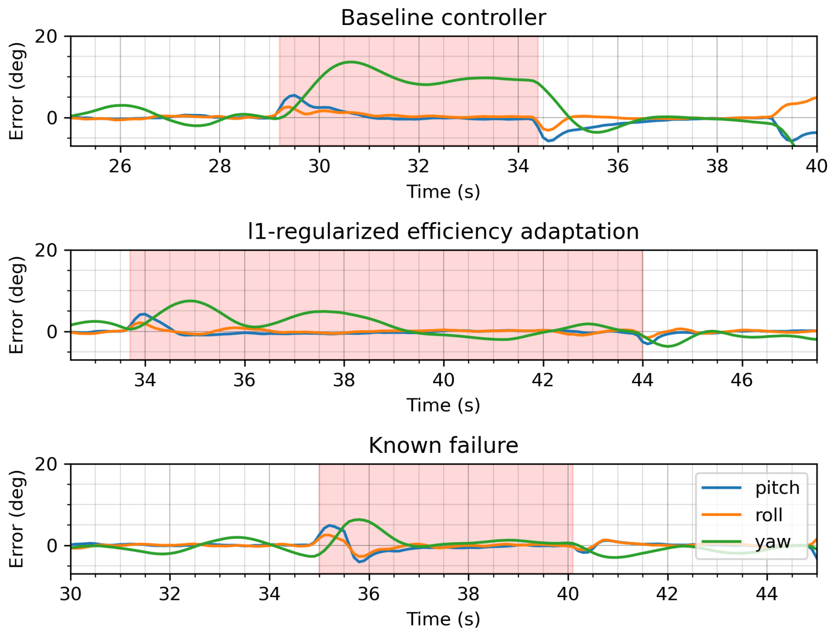


Figure 6.2: **Attitude response given an inboard motor failure.** (top) Baseline ArduCopter response, (middle) adapting to an unknown failure using the $\ell 1$-regularized motor efficiency adaptation given in (6.86) and the allocation algorithm given in (6.13) and (6.14), and (bottom) adapting to a known failure with updated control allocation computed online using (6.13) and (6.14). The $\ell 1$-regularized efficiency adaptation results correspond to the first failure in Fig. 6.3.

matrix at 10 Hz (Figure 6.1, right). A second LUA script running on the Cube Orange+ is used to trigger motor failures, and can take inputs from either ROS or the R/C transmitter to trigger the motor failure.

This architecture allows for the control allocation matrix to be updated in-flight without overburdening the ArduCopter flight controller. On the NanoPi, the efficiency adaptation algorithm incorporates new measurements in under 1 ms. The computation of the current estimate of the control actuation matrix takes less than 5 ms, and resolving the control allocation matrix takes less than 5 ms. This enables measurements to be incorporated essentially as fast as the raw sensor data is available, and the control allocation matrix is updated at 10 Hz, which is about as fast as the ArduCopter flight controller can accept updates.

Experimental performance

Neural-Fly for Fault Tolerance was tested at the North Field which is part of Caltech's Athletics and Recreation Facilities. The North Field is a turf field that measures approximately 150 m by 100 m, allowing for safe flight testing of the aircraft.

Figure 6.2 shows the attitude response of the aircraft to an inboard motor failure at hover. The baseline ArduCopter response does not include any adaptation algorithm and relies on the built-in controller to stabilize the aircraft. The $\ell 1$-regularized efficiency adaptation method uses the sparse failure identification method in (6.86) and the control allocation algorithm in (6.13) and (6.14) to rebalance the system in real time. The known failure case uses the same control allocation algorithm, but directly senses the motor failure.

The known failure case demonstrates the ideal system response to a sudden motor failure. Although the failure is perfectly known, it is not known a priori, and the motors take time to spin up to the desired speed. The $\ell 1$-regularized efficiency adaptation method is able to recover the desired roll and pitch response as quickly as the known failure case, but the yaw response is slower, and it maintains similar maximum roll, pitch, and yaw errors. The baseline ArduCopter response is able to recover the roll and pitch, but more slowly than the other two methods, and it does not recover the desired yaw.

Figure 6.3 shows the motor efficiency estimates for the same experiment, and includes four back to back inboard motor failures. The adaptation algorithm is able to correct for the failures almost instantly. The adaptation algorithm is also able to forget the failures that have been corrected, but it takes a few seconds to do so. During the third failure where motor 5 is turned off, the wrong motor is initially identified as the failed motor, but the algorithm corrects itself. Furthermore, the system still recovers the desired attitude response, even though the wrong motor is identified as the failed motor. Future tests will reset the adaptation algorithm between tests to ensure that the tests are independent.

6.5 Conclusion

In this research, we set out to develop a method for detecting and compensating for motor failures in multirotor UAVs. Our proposed sparse failure identification method has shown promising results, successfully isolating a failed motor within one second and rebalancing the system even when the estimated efficiency does not reach the true value. We also introduced the maximum-control-authority allocation scheme which has shown significant improvements over previous work, particularly in maximizing and maintaining control authority in the presence of a fault.

However, the proposed method's performance in the face of multiple motor failures requires further exploration. Another area for improvement is incorporating a

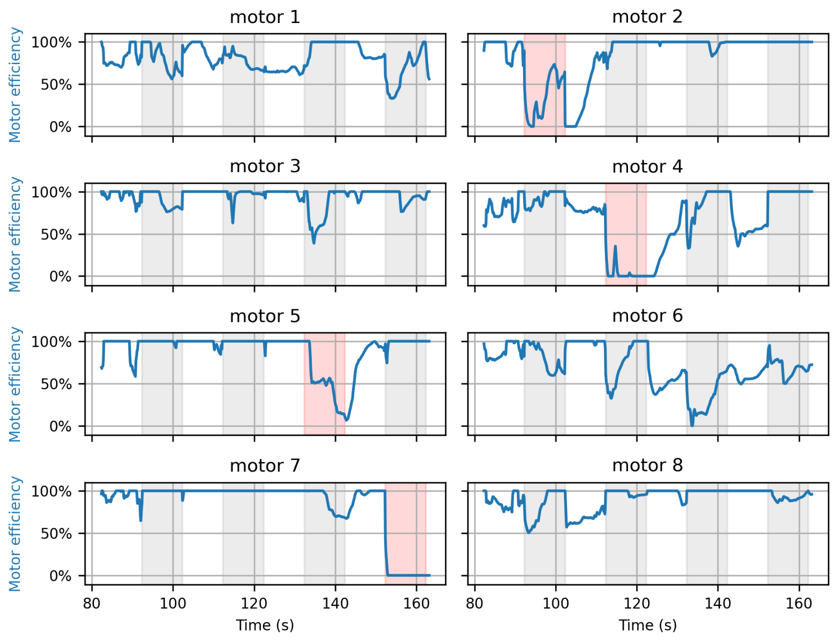


Figure 6.3: **Estimated motor efficiencies.** The $\ell 1$-regularized efficiency adaptation method is able to correctly isolate the failures instantly in some cases and after 5-10 s for other cases. The adaptation takes a few seconds to forget failures that have been corrected.

learned model of the dynamics to more accurately estimate the efficiency values by reducing the effects of unmodeled dynamics.

Despite these areas for further investigation, our research represents a significant step forward in developing robust fault detection and compensation strategies for UAVs. By enhancing the safety and reliability of UAV operations, our work holds potential to contribute to a wide range of applications, from emergency response to last-mile delivery.

Future work will not only address the limitations outlined but will also focus on dynamic testing of the proposed method, incorporating more aggressive maneuvers and higher speeds. The ultimate goal is to continue paving the way towards more resilient, adaptable, and safe UAV systems.

Acknowledgements

I would like to thank Dr. Matthew Anderson and Joshua Cho for their help in this work, especially for their work setting up and conducting hardware experiments, and Joshua Cho for providing the diagram included in Fig. 6.1.

References

[1] G. K. Fourlas and G. C. Karras, "A survey on fault diagnosis methods for uavs," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2021, pp. 394–403. DOI: 10.1109/ICUAS51884.2021.9476733.

[2] R. Puchalski and W. Giernacki, "UAV fault detection methods, state-of-the-art," *Drones*, vol. 6, no. 11, p. 330, 11 Nov. 2022, ISSN: 2504-446X. DOI: 10.3390/drones6110330. [Online]. Available: https://www.mdpi.com/2504-446X/6/11/330 (visited on 02/23/2023).

[3] M. W. Oppenheimer, D. B. Doman, and M. A. Bolender, "Control allocation for over-actuated systems," in *2006 14th Mediterranean Conference on Control and Automation*, Jun. 2006, pp. 1–6. DOI: 10.1109/MED.2006.328750.

[4] X. Shi, K. Kim, S. Rahili, and S.-J. Chung, "Nonlinear control of autonomous flying cars with wings and distributed electric propulsion," in *2018 IEEE Conference on Decision and Control (CDC)*, Miami Beach, FL: IEEE, Dec. 2018, pp. 5326–5333, ISBN: 978-1-5386-1395-5. DOI: 10.1109/CDC.2018.8619578. [Online]. Available: https://ieeexplore.ieee.org/document/8619578/ (visited on 08/25/2020).

[5] K. Kim, S. Rahili, X. Shi, S.-J. Chung, and M. Gharib, "Controllability and design of unmanned multirotor aircraft robust to rotor failure," in *AIAA Scitech 2019 Forum*, San Diego, California: American Institute of Aeronautics and Astronautics, Jan. 7, 2019, ISBN: 978-1-62410-578-4. DOI: 10.2514/6.2019-1787. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2019-1787 (visited on 01/30/2023).

[6] T. A. Johansen and T. I. Fossen, "Control allocation—a survey," *Automatica*, vol. 49, no. 5, pp. 1087–1103, May 1, 2013, ISSN: 0005-1098. DOI: 10.1016/j.automatica.2013.01.035. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109813000368 (visited on 03/30/2021).

[7] O. Harkegard, "Efficient active set algorithms for solving constrained least squares problems in aircraft control allocation," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 2, Las Vegas, NV, USA: IEEE, 2002, pp. 1295–1300, ISBN: 978-0-7803-7516-1. DOI: 10.1109/CDC.2002.1184694. [Online]. Available: http://ieeexplore.ieee.org/document/1184694/ (visited on 09/13/2022).

[8] S. Fuhrer, S. Verling, T. Stastny, and R. Siegwart, "Fault-tolerant flight control of a VTOL tailsitter UAV," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 4134–4140. DOI: 10.1109/ICRA.2019.8793467.

[9] C. Rago, R. Prasanth, R. Mehra, and R. Fortenbaugh, "Failure detection and identification and fault tolerant control using the IMM-KF with applications

to the eagle-eye UAV," in *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, vol. 4, Dec. 1998, 4208–4213 vol.4. DOI: 10.1109/CDC.1998.761963.

[10] H. Blom and Y. Bar-Shalom, "The interacting multiple model algorithm for systems with Markovian switching coefficients," *IEEE Transactions on Automatic Control*, vol. 33, no. 8, pp. 780–783, Aug./1988, ISSN: 00189286. DOI: 10.1109/9.1299. [Online]. Available: http://ieeexplore.ieee.org/document/1299/ (visited on 03/30/2023).

[11] M. Saied, H. Shraim, C. Francis, I. Fantoni, and B. Lussier, "Actuator fault diagnosis in an octorotor UAV using sliding modes technique: Theory and experimentation," in *2015 European Control Conference (ECC)*, Jul. 2015, pp. 1639–1644. DOI: 10.1109/ECC.2015.7330772.

[12] A. Freddi, S. Longhi, and A. Monteriù, "Actuator fault detection system for a mini-quadrotor," in *2010 IEEE International Symposium on Industrial Electronics*, Jul. 2010, pp. 2055–2060. DOI: 10.1109/ISIE.2010.5637750.

[13] D. Rotondo, A. Cristofaro, T. A. Johansen, F. Nejjari, and V. Puig, "Detection of icing and actuators faults in the longitudinal dynamics of small UAVs using an LPV proportional integral unknown input observer," in *2016 3rd Conference on Control and Fault-Tolerant Systems (SysTol)*, Sep. 2016, pp. 690–697. DOI: 10.1109/SYSTOL.2016.7739829.

[14] A. Vahidi, A. Stefanopoulou, and H. Peng, "Recursive least squares with forgetting for online estimation of vehicle mass and road grade: Theory and experiments," *Vehicle System Dynamics*, vol. 43, no. 1, pp. 31–55, Jan. 2005, ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423110412331290446. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00423110412331290446 (visited on 12/08/2022).

[15] G. Chowdhary and E. Johnson, "Concurrent learning for convergence in adaptive control without persistency of excitation," in *49th IEEE Conference on Decision and Control (CDC)*, Dec. 2010, pp. 3674–3679. DOI: 10.1109/CDC.2010.5717148.

[16] E. N. Johnson and A. J. Calise, "Limited authority adaptive flight control for reusable launch vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 26, no. 6, pp. 906–913, Nov. 2003, ISSN: 0731-5090, 1533-3884. DOI: 10.2514/2.6934. [Online]. Available: https://arc.aiaa.org/doi/10.2514/2.6934 (visited on 10/10/2021).

[17] M. O'Connell, G. Shi, X. Shi, *et al.*, "Neural-Fly enables rapid learning for agile flight in strong winds," *Science Robotics*, May 4, 2022. DOI: 10.1126/scirobotics.abm6597. [Online]. Available: https://www.science.org/doi/full/10.1126/scirobotics.abm6597 (visited on 05/13/2022),

[18] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, Apr. 2018, ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2776353.

[19] N. Hovakimyan, E. Lavretsky, and C. Cao, "Adaptive dynamic inversion via time-scale separation," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec. 2006, pp. 1075–1080. DOI: 10.1109/CDC.2006.377287.

[20] S. Mallikarjunan, B. Nesbitt, E. Kharisov, E. Xargay, N. Hovakimyan, and C. Cao, "L1 adaptive controller for attitude control of multirotors," in *AIAA Guidance, Navigation, and Control Conference*, Minneapolis, Minnesota: American Institute of Aeronautics and Astronautics, Aug. 13, 2012, ISBN: 978-1-60086-938-9. DOI: 10.2514/6.2012-4831. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2012-4831 (visited on 03/04/2022).

[21] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear MPC for quadrotors," Sep. 9, 2021. arXiv: 2109.04210 [cs]. [Online]. Available: http://arxiv.org/abs/2109.04210 (visited on 09/16/2021).

[22] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "L1-adaptive MPPI architecture for robust and agile control of multirotors," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 7661–7666. DOI: 10.1109/IROS45743.2020.9341154.

[23] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," in *2018 IEEE Conference on Decision and Control (CDC)*, Dec. 2018, pp. 4282–4288. DOI: 10.1109/CDC.2018.8619621.

[24] G. Shi, X. Shi, M. O'Connell, *et al.*, "Neural lander: Stable drone landing control using learned dynamics," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 9784–9790. DOI: 10.1109/ICRA.2019.8794351,

[25] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[26] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[27] H. K. Khalil, *Nonlinear Systems, 3rd Edition*. Prentice Hall, 2002. [Online]. Available: https://www.pearson.com/content/one-dot-com/one-dot-com/us/en/higher-education/program.html (visited on 09/02/2021).

[28] E. Tang, P. Spieler, M. Anderson, and S.-J. Chung, “Design of the next-generation autonomous flying ambulance,” in *AIAA Scitech 2021 Forum*, VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Jan. 11, 2021, ISBN: 978-1-62410-609-5. DOI: 10.2514/6.2021-1514. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2021-1514 (visited on 02/23/2023).

[29] ArduPilot. “ArduPilot,” ArduPilot. (), [Online]. Available: https://ardupilot.org/.

Chapter 7

CONCLUSION

This thesis has addressed numerous challenges to ensure safe and reliable learning-based control methods. Black-box algorithms can be difficult to verify under all possible conditions; however, by combining a black-box algorithm with classical control methods, the methods presented here not only demonstrate safe and reliable learning-based control, but also outperform state-of-the-art algorithms in agile flight control. The key challenges addressed in this thesis include overcoming unmodeled disturbances, adapting to varying exogenous disturbances, compensating for delays, and fault-tolerant control.

In Chapter 2, we outlined the fundamental considerations for learning residual dynamics in a reliable control architecture. This led to three key insights. First, spectral normalization ensures the existence of a control solution and stability of the control equations. Second, the nonlinear control equations presented by the neural network can be solved quickly and accurately using a fixed point iteration scheme. Third, we proved the robustness of the system, demonstrating that the tracking error scales linearly with the learning error.

In Chapters 3 and 4, we extended the residual-learning-based control framework to online-learning of exogenous disturbances. This work built on the idea of linearly separating the dependence on exogenous and endogenous factors. The endogenous dependence was represented by a neural network and trained offline using DAIML, a novel formulation of meta-learning. This learning algorithm was designed to decouple the correlation between the training data and training conditions, allowing effective and non-overfit modeling of the endogenous dependence. The neural network was updated in real-time and incorporated into the control architecture using a robust adaptive control scheme. This adaptive control scheme does not require persistent excitation of the neural network. The robustness of Neural-Fly was thoroughly studied, showing that the tracking error is linearly bounded by the representation error of the neural network. Neural-Fly outperformed state-of-the-art control algorithms when implemented on the same hardware and tested in various wind conditions, demonstrating that robust and reliable online-learning-based control is achievable.

In Chapter 5, we proposed a first-order delay compensation for handling thrust and attitude control delays. The residual-learning-based control schemes considered in this thesis achieve their robustness, in part, by limiting the scope of the learning problem. Thus, the residual learning is unable to correct for system time-delays. This was a key motivation for studying this first-order delay compensation approach. In quadrotor simulation results, this method was shown to maintain nominal control performance even as dead time and motor response delays increased to 100 ms. This method is lightweight and adds minimal overhead, allowing for easy incorporation into existing control architectures, such as the residual-learning-based control frameworks considered in this thesis. Further work is needed to estimate varying delay times in real systems and to understand noise amplification limits of this method.

In Chapter 6, we studied methods to extend residual-learning-based control schemes from Chapters 2 to 4 to fault-tolerant control by adapting the control at the mixer level. The previous methods are limited in how they can respond to faults, as they only apply a residual correction to the system. By adapting the control at the mixer level, we can extend residual-learning-based control to handle a wider range of disturbances, including actuator faults. Further work is needed to accurately isolate multiple simultaneous faults.

The proposed residual-learning-based control methods have the potential to significantly improve the performance, reliability, and safety of UAVs in various real-world applications, such as delivery services, search and rescue, and urban air mobility. Chapters 2 to 4 demonstrate reliable ways to incorporate learning into controlsystems today, allowing for more agile and precise control. Chapters 5 and 6 address fundamental limitations in the types of disturbances that can be corrected for by residual-learning-based control schemes.

Future research should explore the use of residual-learning-based control schemes in more complex systems, such as VTOL aircraft and legged robots. Further research is needed to address the limitations of the methods in Chapters 5 and 6. Future work should also explore the use of residual-learning-based control schemes in motion planning and state estimation.

The work presented in this thesis represents a significant step forward in the development of safe and reliable learning-based control methods. Residual learning can correct for unmodeled disturbances reliably, and the methods presented here can be incorporated into existing control architectures to enable more precise and agile control. As UAV technology continues to evolve and learning algorithms continue to progress, the insights and methods presented in this thesis will serve as both a demonstration of the potential of learning-based control and a foundation for key considerations in the development of safe and reliable learning-based control systems.