Neural-Fly Enables Rapid Learning for Agile Flight in Strong Winds

Michael O'Connell†, Guanya Shi†, Xichen Shi,
Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, Soon-Jo Chung*

Division of Engineering and Applied Science, California Institute of Technology
† The first two authors contributed equally to this article. Alphabetical order.
* Corresponding authors. Email: sjchung@caltech.edu

This is the accepted version of Science Robotics Vol. 7, Issue 66, eabm6597 (2022)
DOI: 10.1126/scirobotics.abm6597 Video: https://youtu.be/TuF9teCZX0U
Data and training code: https://github.com/aerorobotics/neural-fly

Abstract

Executing safe and precise flight maneuvers in dynamic high-speed winds is important for the ongoing commoditization of uninhabited aerial vehicles (UAVs). However, since the relationship between various wind conditions and its effect on aircraft maneuverability is not well understood, it is challenging to design effective robot controllers using traditional control design methods. We present Neural-Fly, a learning-based approach that allows rapid online adaptation by incorporating pre-trained representations through deep learning. Neural-Fly builds on two key observations that aerodynamics in different wind conditions share a common representation and that the wind-specific part lies in a low-dimensional space. To that end, Neural-Fly uses a proposed learning algorithm, Domain Adversarially Invariant Meta-Learning (DAIML), to learn the shared representation, only using 12 minutes of flight data. With the learned representation as a basis, Neural-Fly then uses a composite adaptation law to update a set of linear coefficients for mixing the basis elements. When evaluated under challenging wind conditions generated with the Caltech Real Weather Wind Tunnel with wind speeds up to $43.6\,\mathrm{km/h}$ ($12.1\,\mathrm{m/s}$), Neural-Fly achieves precise flight control with substantially smaller tracking error than state-of-the-art nonlinear and adaptive controllers. In addition to strong empirical performance, the exponential stability of Neural-Fly results in robustness guarantees. Finally, our control design extrapolates to unseen wind conditions, is shown to be effective for outdoor flights with only on-board sensors, and can transfer across drones with minimal performance degradation.

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.

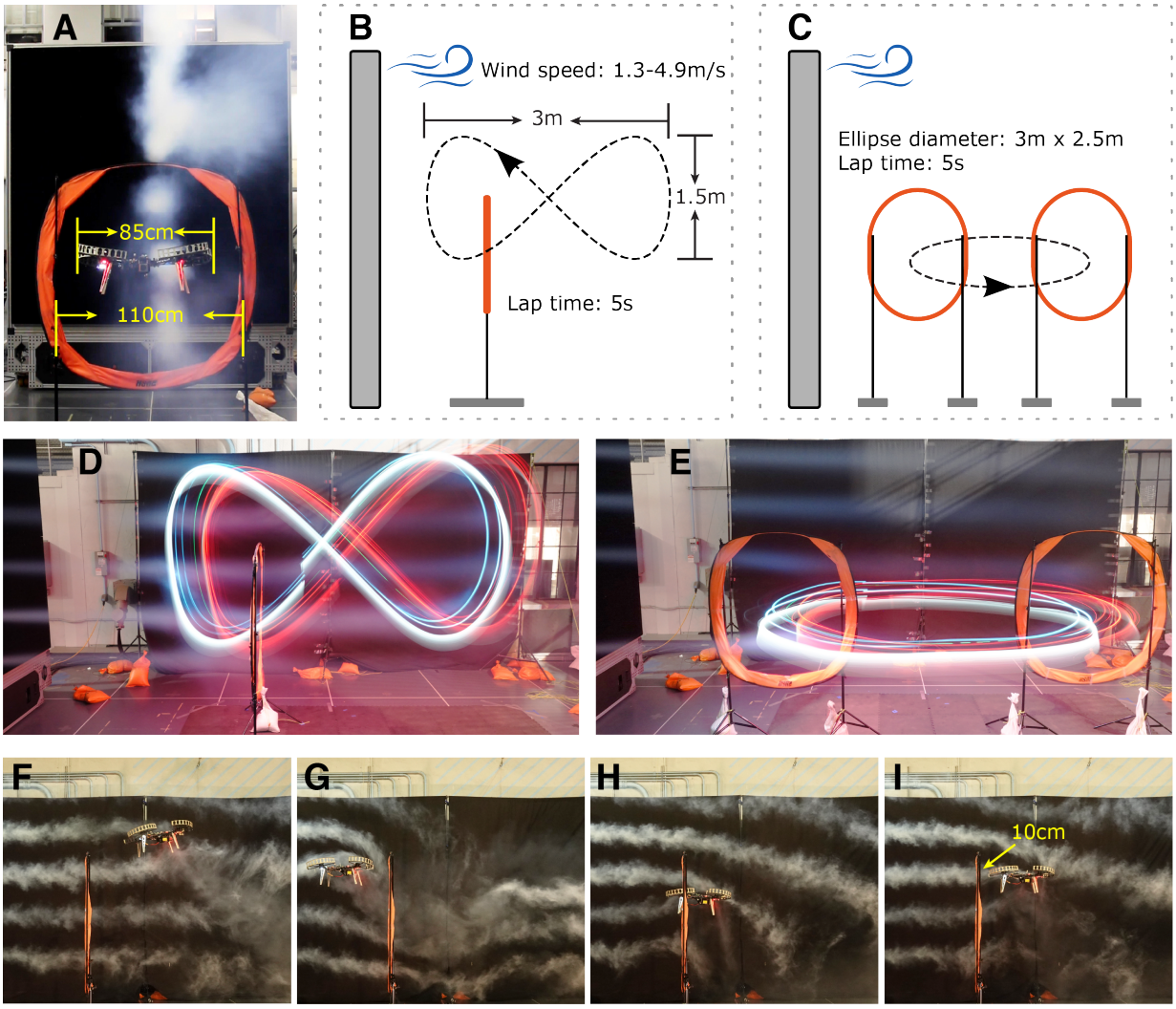

Figure 1: **Agile flight through narrow gates.** (**A**) Caltech Real Weather Wind Tunnel system, the quadrotor UAV, and the gate. In our flight tests, the UAV follows an agile trajectory through narrow gates, which are slightly wider than the UAV itself, under challenging wind conditions. (**B-C**)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

Unmodeled and often complex aerodynamics are among the most notable challenges to precise flight control. Flying in windy environments (as shown in Fig. 1) introduces even more complexity because of the unsteady aerodynamic interactions between the drone, the induced airflow, and the wind (see Fig. 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, 5, 6, 7]. In parallel, there has been increased interest in data-driven modeling of aerodynamics (e.g., [8, 9, 10, 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-changing wind conditions. Our method, depicted in Fig. 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 use of composite adaptive control was inspired by [13, 14]. 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

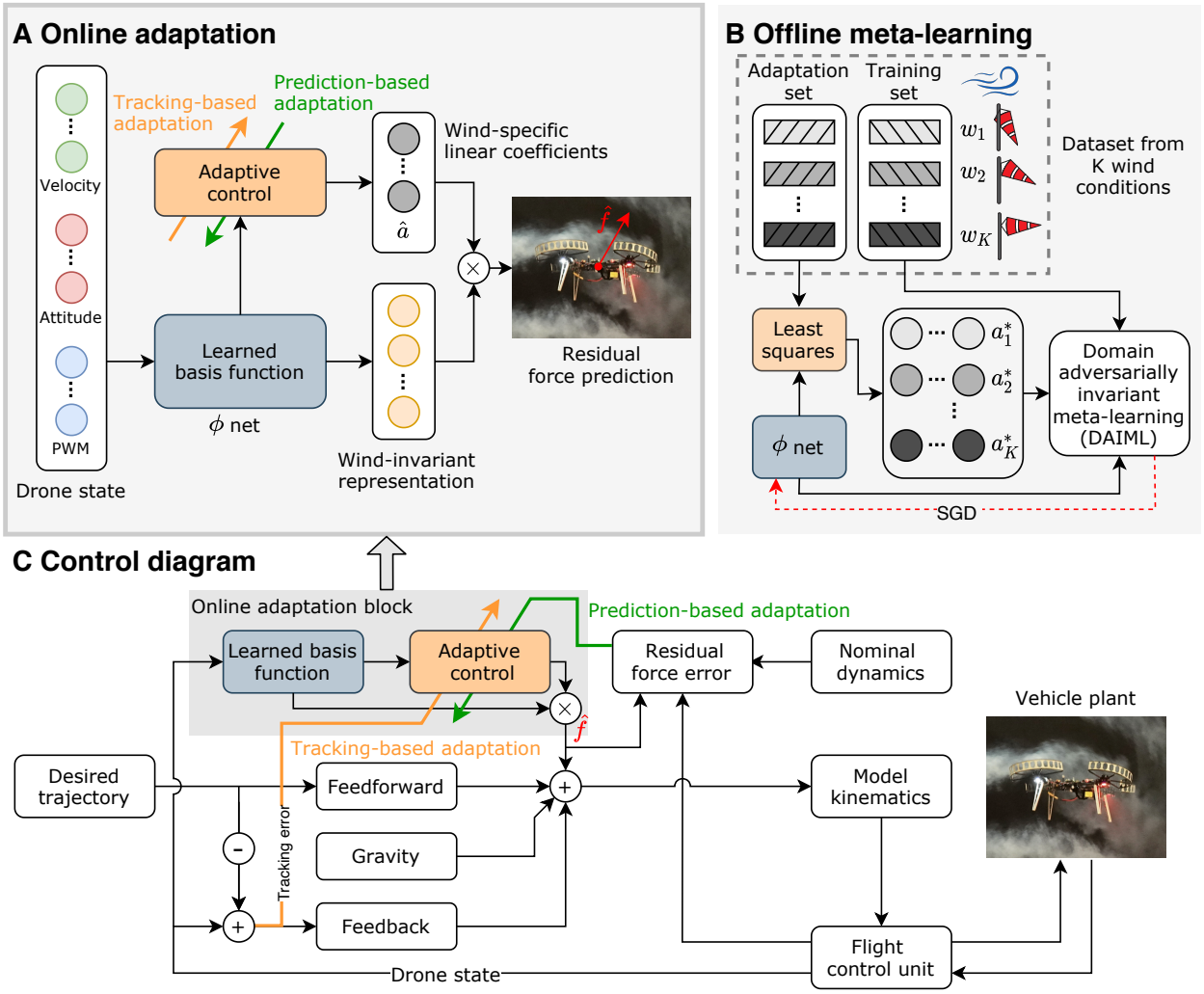

Figure 2: **Offline meta-learning and online adaptive control design.** (**A**) The online adaptation block in our adaptive controller. Our controller leverages the meta-trained basis function ϕ, which is a wind-invariant representation of the aerodynamic effects, and uses composite adaptation (that 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 grey part corresponds to (A). Interpreting the learned block as an aerodynamic force allows it to be incorporated into the feedback control easily.

default regulation 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. 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 [15]. 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 [16], 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 [4, 7, 11, 2]. However, those approaches require specialized onboard hardware [4], full custom flight control stacks [4, 7], or cannot adapt to external wind disturbances [11, 2]. 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 $6.6\,\mathrm{cm}$ and drone ground speeds up to $12.9\,\mathrm{m/s}$ [4]. However, [4] relies on high-frequency control updates ($500\,\mathrm{Hz}$) 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 $\sim 6\,\mathrm{cm}$ tracking error [2], uses an attitude controller running at $4000\,\mathrm{Hz}$, 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 [13, 17, 18, 19, 20, 21]. 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 [13]. The idea of composite adaptation was first introduced in [14].

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 [22]. 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 [23, 24], 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 of 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 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 [20, 25, 26, 27, 28]. A recent work applies this idea to perform an impressive quadrotor flip [29]. 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 [30]. 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 [31, 32, 33]. 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 [34, 35], changing suspended payload for drones [36], and unknown operating conditions for wheeled robots [37].

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 [31, 34, 36, 38]. 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., $\sim$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 [39, 37, 40, 41]. The intuition is that, different environments share a common representation (e.g., the wind-invariant representation in Fig. 2(A)), and the environment-specific part is in a low-dimensional space (e.g., the wind-specific linear weight in Fig. 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 [39], later followed by [40]. However, the representation learned in [39] is ineffective and the tracking performance in [39] is similar as the baselines; [40] 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 a 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 [42, 43, 44]. 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 [42] 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.

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.

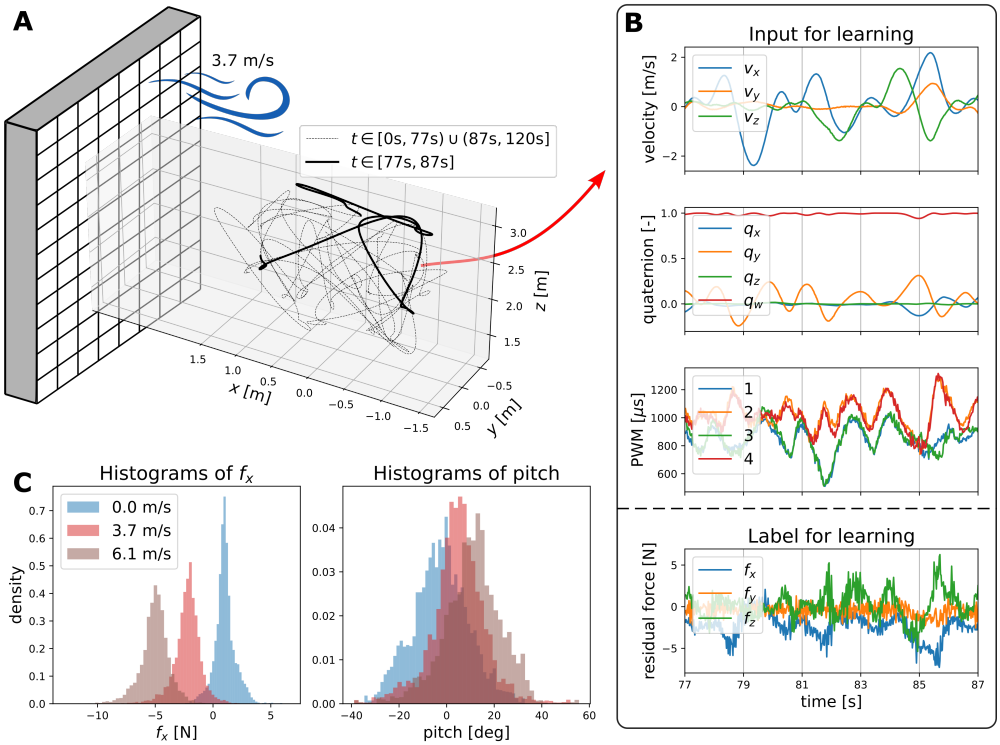

Figure 3: **Training data collection.** (**A**) The xyz position along a two-minute randomized trajectory for data collection with wind speed $8.3\,\mathrm{km/h}$ ($3.7\,\mathrm{m/s}$), in the Caltech Real Weather Wind Tunnel. (**B**) A typical 10-second trajectory of the inputs (velocity, attitude quaternion, 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 $50\,\mathrm{Hz}$, a WiFi 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 $43.6\,\mathrm{km/h}$ 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. 1(A). The quadrotor weighs $2.6\,\mathrm{kg}$ 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 [15]. The UAV incorporates a Raspberry Pi 4 onboard computer running a Linux operation 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 $85\,\mathrm{cm}$ in width, $75\,\mathrm{cm}$ in length, and $93\,\mathrm{cm}$ 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 (Section S1) 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 $50\,\mathrm{Hz}$. 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 $1.4\,\mathrm{kg}$) has a symmetric X configuration, $52\,\mathrm{cm}$ in width and $52\,\mathrm{cm}$ 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. 1(A)). The wind tunnel is a $3\,\mathrm{m}$ by $3\,\mathrm{m}$ array of 1296 independently controllable fans capable of generating wind conditions up to $43.6\,\mathrm{km/h}$. 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 $0\,\mathrm{km/h}$ and $43.6\,\mathrm{km/h}$ ($12.1\,\mathrm{m/s}$). The second type of wind follows a sinusoidal function in time, e.g., $30.6 + 8.6\sin(t)\ \mathrm{km/h}$. Note that the training data only covers constant wind speeds up to $6.1\,\mathrm{m/s}$. To visualize the wind, we use 5 smoke generators to indicate the direction and intensity of the wind condition (see examples in Fig. 1 and Video 1).

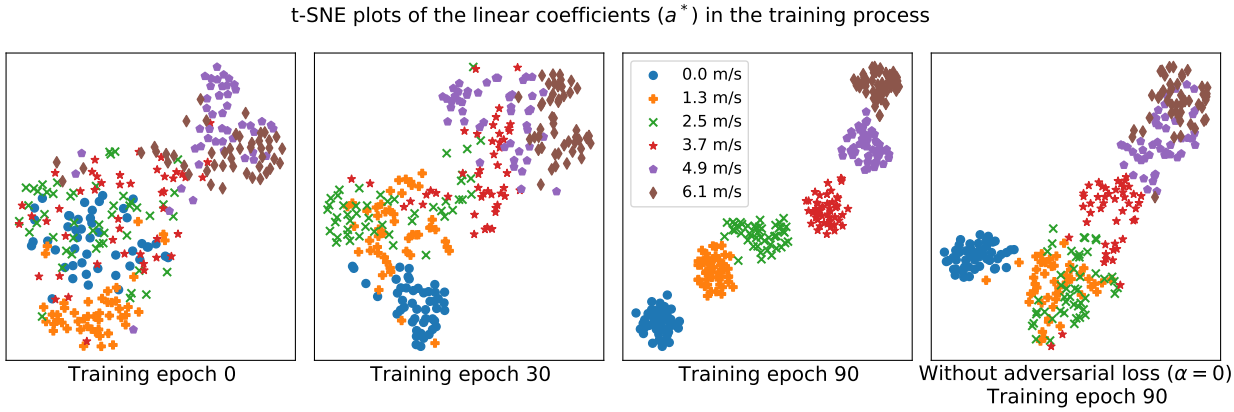

Figure 4: **t-SNE plots showing the evolution of the linear weights (a^*) during the training process.** As the number of training epochs increases, the distribution of a^* becomes more clustered with similar wind speed clusters near each other. The clustering also has a physical meaning: after training 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.

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 $0\,\mathrm{km/h}$ to $22.0\,\mathrm{km/h}$. However, in experiments we used wind speeds up to $43.6\,\mathrm{km/h}$ ($12.1\,\mathrm{m/s}$) to study how our methods extrapolate to unseen wind conditions (e.g., Fig. 6). The data is collected at 50 Hz with a total of $36,000$ data points. Figure 3(A) shows the data collection process, and Fig. 3(B) shows the inputs and labels of the training data, under one wind condition of $13.3\,\mathrm{km/h}$ ($3.7\,\mathrm{m/s}$). Figure 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. 2(B). Figure 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. 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 [16], which is sent to the PX4 flight controller. The three baselines include: globally exponentially-stabilizing nonlinear tracking controller for quadrotor control [45, 8, 46], 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 unmodelled residual force (that is, each baseline method has the same control structure, in Fig. 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. 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 $2.5\,\mathrm{m}$ wide, $1.5\,\mathrm{m}$ tall figure-8 trajectory with a lap time of $6.28\,\mathrm{s}$ under constant, uniform wind speeds of $0\,\mathrm{km/h}$, $15.1\,\mathrm{km/h}$ ($4.2\,\mathrm{m/s}$), $30.6\,\mathrm{km/h}$ ($8.5\,\mathrm{m/s}$), and $43.6\,\mathrm{km/h}$ ($12.1\,\mathrm{m/s}$) and under time-varying wind speeds of $30.6 + 8.6\sin(t)\ \mathrm{km/h}$ ($8.5 + 2.4\sin(t)\ \mathrm{m/s}$).

The flight trajectory for each of the experiments is shown in Fig. 5, which includes a warm up lap and six $6.28\,\mathrm{s}$ 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: $43.6\,\mathrm{km/h}$ 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 1, we tabulate the root-mean-square position error and mean position error values over the six laps for each experiment. Figure 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 Neural-Fly-Transfer outperform the state-of-the-art baseline methods, including the $30.6\,\mathrm{km/h}$, $43.6\,\mathrm{km/h}$, and 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.

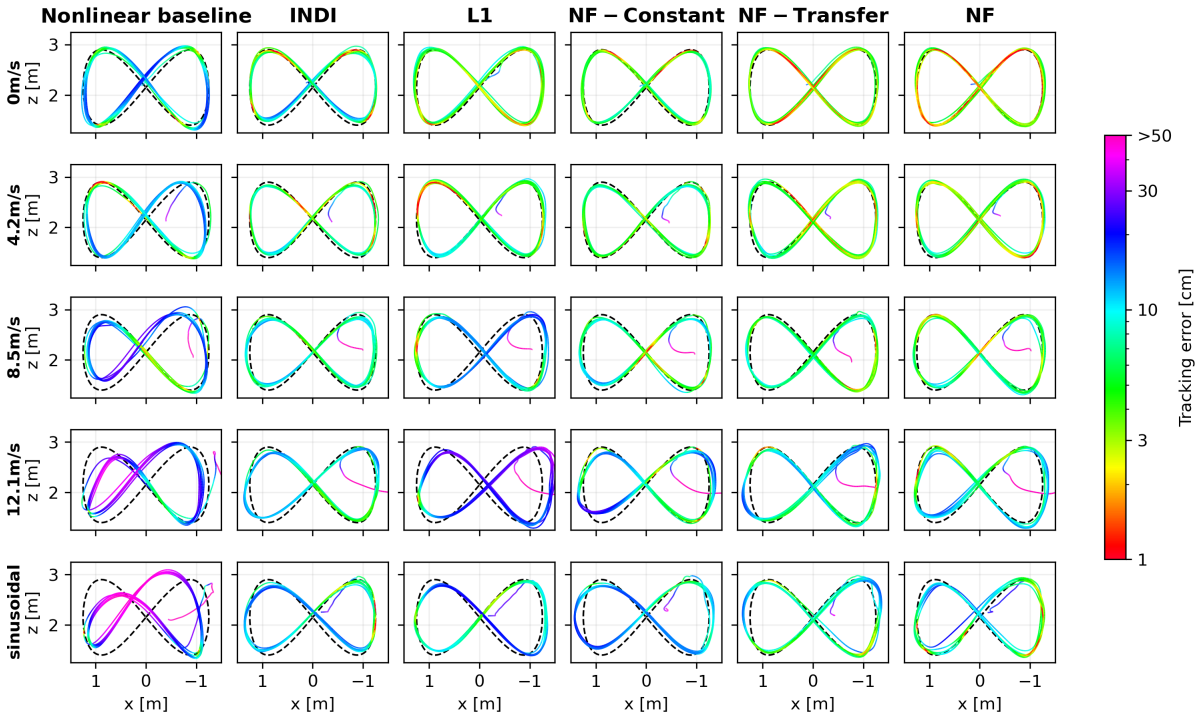

Figure 5: **Depiction of the trajectory tracking performance of each controller in several wind conditions.** The baseline nonlinear controller can track the trajectory well, however, the performance substantially degrades at higher wind speeds. INDI, $\mathcal{L}_1$, and Neural-Fly-Constant have similar 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.

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. 1 and Video 1. As shown in Fig. 1(A), the gates used in our experiments are $110\,\mathrm{cm}$ in width, which is only slightly wider than the drone ($85\,\mathrm{cm}$ wide, $75\,\mathrm{cm}$ 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. 1(B,D,F-I) and Video 1), the desired trajectory is a $3\,\mathrm{m}$ by $1.5\,\mathrm{m}$ figure-8 in the $x-z$ plane with a lap time of $5\,\mathrm{s}$. A gate is

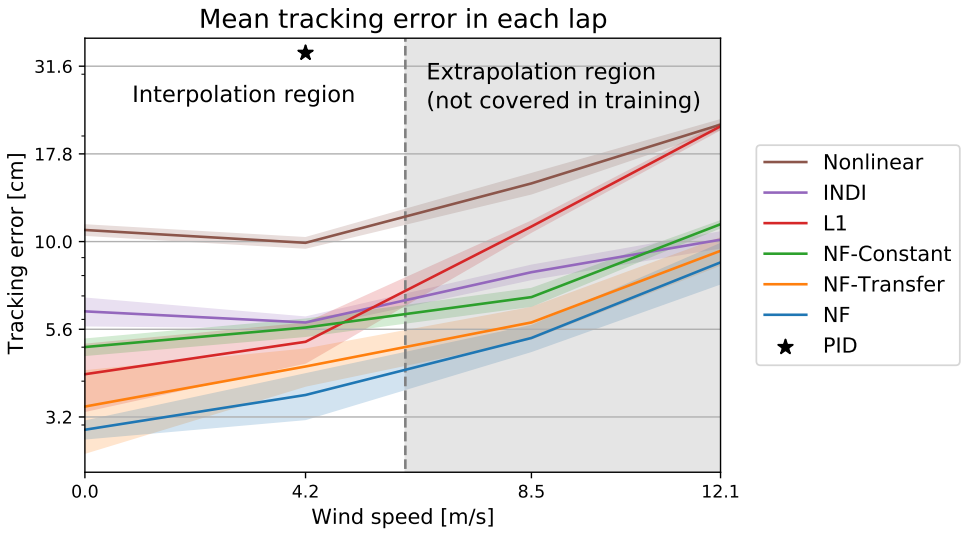

Figure 6: **Mean tracking errors of each lap in different wind conditions.** This figure shows position tracking errors of different methods as wind speed increases. Solid lines show the mean error over 6 laps and the shade areas show standard deviation of the mean error on each lap. The grey area indicates 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.

placed at the left bottom part of the trajectory. The minimum clearance is about $10\,\mathrm{cm}$ (see Fig. 1(I), which requires that the controller precisely tracks the trajectory. The maximum speed and acceleration of the desired trajectory are $2.7\,\mathrm{m/s}$ and $5.0\,\mathrm{m/s^2}$, respectively. The wind speed is $3.1\,\mathrm{m/s}$. 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. 1(C,E) and Video 1), the desired trajectory is a $3\,\mathrm{m}$ by $2.5\,\mathrm{m}$ ellipse in the $x-y$ plane with a lap time of $5\,\mathrm{s}$. We placed two gates on the left and right sides of the ellipse. As with the first task, the wind speed is $3.1\,\mathrm{m/s}$.

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 1(D,E) are two long-exposure photos with an exposure time of $5\,\mathrm{s}$, 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. 1(A)). Figure 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.

Table 1: Tracking error statistics in cm for different wind conditions. Two metrics are considered: root-mean-square (RMS) and mean.

Method \ Wind speed [m/s]	0		4.2		8.5		12.1		$8.5 + 2.4\sin(t)$	
	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**

Outdoor Experiments

We tested our algorithm outdoors in gentle breeze conditions (wind speeds measured up to $17\,\mathrm{km/h}$). 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 $7.5\,\mathrm{cm}$ mean tracking error, shown in Fig. 7.

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 $43.6\,\mathrm{km/h}$ ($12.1\,\mathrm{m/s}$). Furthermore, we observe a mean tracking error of $2.9\,\mathrm{cm}$ in $0\,\mathrm{km/h}$ 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 measuremnts. 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 $1\,\mathrm{cm}$, which is a practical lower bound for the average tracking error in our system (see more details in the supplementary material, Section S8). This is based on the fact that the difference between the OptiTrack position measurement and the onboard EKF position estimate is around $1\,\mathrm{cm}$.

To achieve a tracking error of $1\,\mathrm{cm}$, 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 $15\,\mathrm{ms}$ and often as much as $30\,\mathrm{ms}$. 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., [46]). Our method

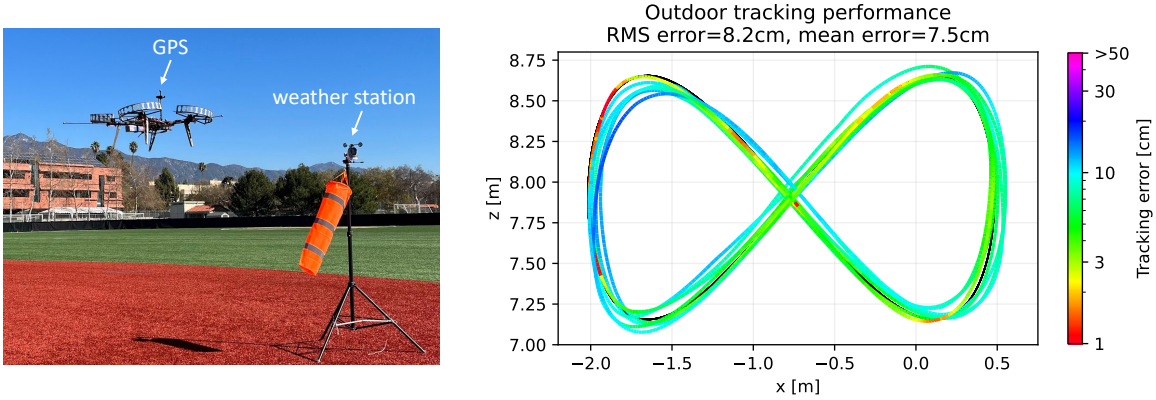

Figure 7: **Outdoor flight setup and performance. Left:** In outdoor experiments, a GPS module is deployed for state estimation, and a weather station records wind profiles. The maximum wind speed during the test was around $17\,\mathrm{km/h}$ ($4.9\,\mathrm{m/s}$). **Right:** Trajectory tracking performance of Neural-Fly.

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.

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 unmodelled dynamics. Testing our method in outdoor flight, the quadrotor maintains precise tracking with only $7.5\,\mathrm{cm}$ tracking error on a gentle breezy day with wind speeds around $17\,\mathrm{km/h}$.

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 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. 2(A)) and the wind condition (i.e., the wind-specific linear weight in Fig. 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 $8.7\,\mathrm{cm}$ in Table 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 $50\,\mathrm{Hz}$, 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 $10\,\mathrm{ms}$ to compute on a compact onboard Linux computer (Raspberry Pi 4). In particular, the feedforward inference time via the learned basis function is about $3.5\,\mathrm{ms}$ and the adaptation update is about $3.0\,\mathrm{ms}$, 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. 1. Note that our method supports any trajectory planners such as [1] or learning-based planners [47, 48, 49]. 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 [22, 50].

4 MATERIALS AND METHODS

Overview

We consider a general robot dynamics model:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = u + f(q,\dot{q},w) \tag{1}$$

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. 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

$$f(q,\dot{q},w) \approx \phi(q,\dot{q})a(w), \tag{2}$$

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 (Section S2), 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. 2(B) and Fig. 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(A) illustrates one trajectory under the wind condition $13.3\,\mathrm{km/h}$ ($3.7\,\mathrm{m/s}$). 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 $0\,\mathrm{km/h}$ to $22.0\,\mathrm{km/h}$ ($6.1\,\mathrm{m/s}$), which are in the white interpolation region in Fig. 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, the 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. 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 Eq. (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

$$D_{w_k} = \left\{ x_k^{(i)}, y_k^{(i)} = f(x_k^{(i)}, w_k) + \epsilon_k^{(i)} \right\}_{i=1}^{N_k} \tag{3}$$

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:

$$\min_{\phi,a_1,\cdots,a_K} \sum_{k=1}^{K}\sum_{i=1}^{N_k} \left\| y_k^{(i)} - \phi(x_k^{(i)})a_k \right\|^2, \tag{4}$$

where $\phi(\cdot) : \mathbb{R}^{2n} \to \mathbb{R}^{n\times h}$ is the representation function and $a_k \in \mathbb{R}^h$ 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 Eq. (4) is a reasonable representation of the unknown dynamics $f(x,w)$.

Domain Shift Problems One challenge of the optimization in Eq. (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(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 Eq. (4) may lead to *over-fitting* and may not properly find a wind-invariant representation ϕ.

To solve the domain shift problem, inspired by [51], we propose the following adversarial optimization framework:

$$\max_{h}\min_{\phi,a_1,\cdots,a_K} \sum_{k=1}^{K}\sum_{i=1}^{N_k} \left(\left\| y_k^{(i)} - \phi(x_k^{(i)})a_k \right\|^2 - \alpha \cdot \text{loss}\left(h(\phi(x_k^{(i)})), k\right)\right), \tag{5}$$

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

$$\text{loss}\left(h(\phi(x_k^{(i)})), k\right) = -\sum_{j=1}^{K} \delta_{kj} \log\left(h(\phi(x_k^{(i)}))^\top e_j\right) \tag{6}$$

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

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 ϕ and h
Result: Trained neural networks ϕ 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 ϕ using stochastic gradient descent (SGD) and spectral normalization with loss

$$\sum_{i \in B} \left(\left\| y_k^{(i)} - \phi(x_k^{(i)}) a^* \right\|^2 - \alpha \cdot \text{loss}\left(h(\phi(x_k^{(i)})), k \right) \right)$$

8 **if** `rand()` $\leq \eta$ **then**
9 Train DNN h using SGD with loss $\sum_{i \in B} \text{loss}\left(h(\phi(x_k^{(i)})), k \right)$
10 **until convergence**

Design of the DAIML Algorithm Finally, we solve the optimization problem in Eq. (5) by the proposed algorithm DAIML (described in Algorithm 1 and illustrated in Fig. 2(B)), which belongs to the category of gradient-based meta-learning [32], but with least squares as the adaptation step. DAIML contains three steps: (i) The adaptation step (Line 4-6) solves an 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 [52].

We further motivate the algorithm design using Fig. 3 and Fig. 4. Figure 3(A,B) shows the input and label from one wind condition, and Fig. 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 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. 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. 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 (Section S3).

Robust Adaptive Controller Design

During the offline meta-training process, a least-squares fit is used to 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 as well as on the tracking error, as illustrated in Fig. 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.

$$u_{\text{NF}} = \underbrace{M(q)\ddot{q}_r + C(q,\dot{q})\dot{q}_r + g(q)}_{\text{nominal model feedforward terms}} \quad \underbrace{-Ks}_{\text{PD feedback}} \quad \underbrace{-\phi(q,\dot{q})\hat{a}}_{\text{learning-based feedforward}} \tag{7}$$

$$\dot{\hat{a}} = \underbrace{-\lambda\hat{a}}_{\text{regularization term}} \quad \underbrace{-P\phi^\top R^{-1}(\phi\hat{a} - y)}_{\text{prediction error term}} \quad \underbrace{+P\phi^\top s}_{\text{tracking error term}} \tag{8}$$

$$\dot{P} = -2\lambda P + Q - P\phi^\top R^{-1}\phi P \tag{9}$$

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. 2. Figure 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 Eq. (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

$$s = \dot{q} - \dot{q}_r = \dot{\tilde{q}} + \Lambda\tilde{q} \tag{10}$$

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 Section S5). 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

$$u_{\text{NL}} = \underbrace{M(q)\ddot{q}_r + C(q,\dot{q})\dot{q}_r + g(q)}_{\text{nonlinear feedforward terms}} \underbrace{-Ks - K_I \int s dt}_{\text{PID feedback}}. \tag{11}$$

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 unmodelled dynamics and disturbance forces.

Our method improves the controller by predicting the unmodeled dynamics and disturbance forces, and, indeed, in Table 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 Eq. (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. 5. This comparison is further discussed in the supplementary materials (Section S7).

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 Eq. (8) and (9), where y is a noisy measurement of f, λ is a damping gain, P is a covariance matrix which evolves according to Eq. (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 [53]. 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 Section S6. 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 Eq. (8) a composite adaptation law, guaranteeing stability of the closed-loop system (see Theorem 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 1. *If we assume that the desired trajectory has bounded derivatives and the system evolves according to the dynamics in Eq.* (1)*, the control law Eq.* (7)*, and the adaptation law Eq.* (8) *and* (9)*, then the position tracking error exponentially converges to the ball*

$$\lim_{t\to\infty} \|\tilde{q}\| \leq \sup_t \left[C_1\|d(t)\| + C_2\|\epsilon(t)\| + C_3\left(\lambda\|a(t)\| + \|\dot{a}(t)\|\right)\right], \tag{12}$$

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

$$\dot{p} = v, \qquad m\dot{v} = mg + Rf_u + f, \tag{13a}$$

$$\dot{R} = RS(\omega), \qquad J\dot{\omega} = J\omega \times \omega + \tau_u, \tag{13b}$$

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 unmodelled aerodynamic effects due to varying wind conditions.

We cast the position dynamics in Eq. (13a) into the form of Eq. (1), by taking $M(q) = mI, C(q, \dot{q}) \equiv 0$, and $u = Rf_u$. Note that the quadrotor attitude dynamics Eq. (13b) is also a special case of Eq. (1) [13, 54], 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. 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 a 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. 2 and 3). The DNN ϕ has four fully-connected hidden layers, with an architecture $11 \to 50 \to 60 \to 50 \to 4$ 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

$$f \approx \begin{bmatrix} \phi(x) & 0 & 0 \\ 0 & \phi(x) & 0 \\ 0 & 0 & \phi(x) \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}, \tag{14}$$

where $a_x, a_y, a_z \in \mathbb{R}^4$ 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 (Section S3).

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., Eq. (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 Section S4. 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 $\hat{f} = -u_{\mathcal{L}_1}$.

SUPPLEMENTARY MATERIALS

REFERENCES

[1] Philipp Foehn, Angel Romero, and Davide Scaramuzza. Time-optimal planning for quadrotor waypoint flight. *Science Robotics*, 6(56), 2021.

[2] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories. *IEEE Robotics and Automation Letters*, 3(2):620–626, April 2018. ISSN 2377-3766. doi: 10.1109/LRA.2017.2776353.

[3] Patricia Ventura Diaz and Steven Yoon. High-fidelity computational aerodynamics of multi-rotor unmanned aerial vehicles. In *2018 AIAA Aerospace Sciences Meeting*, page 1266, 2018.

[4] Ezra Tal and Sertac Karaman. Accurate Tracking of Aggressive Quadrotor Trajectories Using Incremental Nonlinear Dynamic Inversion and Differential Flatness. *IEEE Transactions on Control Systems Technology*, 29(3):1203–1218, May 2021. ISSN 1558-0865. doi: 10.1109/TCST.2020.3001117.

[5] Srinath Mallikarjunan, Bill Nesbitt, Evgeny Kharisov, Enric Xargay, Naira Hovakimyan, and Chengyu Cao. L1 Adaptive Controller for Attitude Control of Multirotors. In *AIAA Guidance, Navigation, and Control Conference*, Minneapolis, Minnesota, August 2012. American Institute of Aeronautics and Astronautics. ISBN 978-1-60086-938-9. doi: 10.2514/6.2012-4831. URL `https://arc.aiaa.org/doi/10.2514/6.2012-4831`.

[6] Jintasit Pravitra, Kasey A. Ackerman, Chengyu Cao, Naira Hovakimyan, and Evangelos 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)*, pages 7661–7666, October 2020. doi: 10.1109/IROS45743.2020.9341154. ISSN: 2153-0866.

[7] Drew Hanover, Philipp Foehn, Sihao Sun, Elia Kaufmann, and Davide Scaramuzza. Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors. *IEEE Robotics and Automation Letters*, 7(2):690–697, 2021.

[8] Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural lander: Stable drone landing control using learned dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9784–9790. IEEE, 2019.

[9] Guanya Shi, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung. Neural-swarm: Decentralized close-proximity multirotor control using learned interactions. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3241–3247. IEEE, 2020.

[10] Guanya Shi, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung. Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions. *IEEE Transactions on Robotics*, 2021.

[11] Guillem Torrente, Elia Kaufmann, Philipp Föhn, and Davide Scaramuzza. Data-driven mpc for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):3769–3776, 2021.

[12] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in Neural Information Processing Systems*, 30:6240–6249, 2017.

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

[14] Jean-Jacques E. Slotine and Weiping Li. Composite adaptive control of robot manipulators. *Automatica*, 25(4):509–519, July 1989. ISSN 0005-1098. doi: 10.1016/0005-1098(89)90094-0. URL https://www.sciencedirect.com/science/article/pii/0005109889900940.

[15] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. PIXHAWK -A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1-2):21–39, August 2012. ISSN 0929-5593. doi: 10.1007/s10514-012-9281-4. URL https://graz.pure.elsevier.com/en/publications/pixhawk-a-micro-aerial-vehicle-design-for-autonomous-flight-using. Publisher: Springer Science + Business Media.

[16] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, May 2011. doi: 10.1109/ICRA.2011.5980409.

[17] Petros A Ioannou and Jing Sun. *Robust adaptive control*, volume 1. Prentice-Hall Upper Saddle River, NJ, 1996.

[18] Miroslav Krstic, Petar V Kokotovic, and Ioannis Kanellakopoulos. *Nonlinear and adaptive control design*. John Wiley & Sons, Inc., 1995.

[19] Kumpati S Narendra and Anuradha M Annaswamy. *Stable adaptive systems*. Courier Corporation, 2012.

[20] Jay A. Farrell and Marios M. Polycarpou. *Adaptive Approximation Based Control*. John Wiley & Sons, Ltd, 2006. ISBN 978-0-471-78181-3. doi: 10.1002/0471781819.fmatter. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/0471781819.fmatter. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471781819.fmatter.

[21] Kevin A Wise, Eugene Lavretsky, and Naira Hovakimyan. Adaptive control of flight: theory, applications, and open problems. In *2006 American Control Conference*, 2006.

[22] Xichen Shi, Patrick Spieler, Ellande Tang, Elena-Sorina Lupu, Phillip Tokumaru, and Soon-Jo Chung. Adaptive Nonlinear Control of Fixed-Wing VTOL with Airflow Vector Sensing. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5321–5327, Paris, France, May 2020. IEEE. ISBN 978-1-72817-395-5. doi: 10.1109/ICRA40945.2020.9197344. URL https://ieeexplore.ieee.org/document/9197344/.

[23] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pages 1177–1184, 2007.

[24] Sahin Lale, Kamyar Azizzadenesheli, Babak Hassibi, and Anima Anandkumar. Model Learning Predictive Control in Nonlinear Dynamical Systems. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 757–762, December 2021. doi: 10.1109/CDC45484.2021.9683670. ISSN: 2576-2370.

[25] J. Nakanishi, J.A. Farrell, and S. Schaal. A locally weighted learning composite adaptive controller with structure adaptation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 882–889 vol.1, September 2002. doi: 10.1109/IRDS.2002.1041502.

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

[27] Eric N Johnson and Anthony J Calise. Limited authority adaptive flight control for reusable launch vehicles. *Journal of Guidance, Control, and Dynamics*, 26(6):906–913, 2003.

[28] Kumpati S Narendra and Snehasis Mukhopadhyay. Adaptive control using neural networks and approximate models. *IEEE Transactions on Neural Networks*, 8(3):475–485, 1997.

[29] Mahdis Bisheban and Taeyoung Lee. Geometric Adaptive Control With Neural Networks for a Quadrotor in Wind Fields. *IEEE Transactions on Control Systems Technology*, 29(4):1533–1548, July 2021. ISSN 1558-0865. doi: 10.1109/TCST.2020.3006184.

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[31] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[32] Timothy M Hospedales, Antreas Antoniou, Paul Micaelli, and Amos J. Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. doi: 10.1109/TPAMI.2021.3079209.

[33] Guanya Shi, Kamyar Azizzadenesheli, Michael O'Connell, Soon-Jo Chung, and Yisong Yue. Meta-adaptive nonlinear control: Theory and algorithms. *Advances in Neural Information Processing Systems*, 34, 2021.

[34] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[35] Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. Rapidly adaptable legged robots via evolutionary meta-learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3769–3776. IEEE, 2020.

[36] Suneel Belkhale, Rachel Li, Gregory Kahn, Rowan McAllister, Roberto Calandra, and Sergey Levine. Model-based meta-reinforcement learning for flight with suspended payloads. *IEEE Robotics and Automation Letters*, 6(2):1471–1478, 2021.

[37] Christopher D McKinnon and Angela P Schoellig. Meta learning with paired forward and inverse models for efficient receding horizon control. *IEEE Robotics and Automation Letters*, 6(2):3240–3247, 2021.

[38] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, pages 617–629. PMLR, 2018.

[39] Michael O'Connell, Guanya Shi, Xichen Shi, and Soon-Jo Chung. Meta-learning-based robust adaptive flight control under uncertain wind conditions. *arXiv preprint arXiv:2103.01932*, 2021.

[40] Spencer M Richards, Navid Azizan, Jean-Jacques E Slotine, and Marco Pavone. Adaptive-control-oriented meta-learning for nonlinear systems. *arXiv preprint arXiv:2103.04490*, 2021.

[41] Matt Peng, Banghua Zhu, and Jiantao Jiao. Linear representation meta-reinforcement learning for instant adaptation. *arXiv preprint arXiv:2101.04750*, 2021.

[42] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

[43] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter 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)*, pages 23–30. IEEE, 2017.

[44] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv preprint arXiv:1906.01728*, 2019.

[45] Daniel Morgan, Giri P Subramanian, Soon-Jo Chung, and Fred Y Hadaegh. Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming. *The International Journal of Robotics Research*, 35(10):1261–1285, 2016.

[46] Xichen Shi, Kyunam Kim, Salar Rahili, and Soon-Jo Chung. Nonlinear control of autonomous flying cars with wings and distributed electric propulsion. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 5326–5333. IEEE, 2018.

[47] Yashwanth Kumar Nakka, Anqi Liu, Guanya Shi, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. Chance-constrained trajectory optimization for safe exploration and learning of nonlinear systems. *IEEE Robotics and Automation Letters*, 6(2):389–396, 2020.

[48] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021. doi: 10.1126/scirobotics.abg5810. URL `https://www.science.org/doi/abs/10.1126/scirobotics.abg5810`.

[49] Anqi Liu, Guanya Shi, Soon-Jo Chung, Anima Anandkumar, and Yisong Yue. Robust regression for safe exploration in control. In *Learning for Dynamics and Control*, pages 608–619. PMLR, 2020.

[50] Kyunam Kim, Patrick Spieler, Elena-Sorina Lupu, Alireza Ramezani, and Soon-Jo Chung. A bipedal walking robot that can fly, slackline, and skateboard. *Science Robotics*, 6(59):eabf8136, 2021. doi: 10.1126/scirobotics.abf8136.

[51] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[52] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 27, 2014.

[53] R. E. Kalman and R. S. Bucy. New Results in Linear Filtering and Prediction Theory. *Journal of Basic Engineering*, 83(1):95–108, March 1961. ISSN 0021-9223. doi: 10.1115/1.3658902. URL https://asmedigitalcollection.asme.org/fluidsengineering/article/83/1/95/426820/New-Results-in-Linear-Filtering-and-Prediction.

[54] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1 edition, December 2017. ISBN 978-1-315-13637-0. doi: 10.1201/9781315136370. URL https://www.taylorfrancis.com/books/9781351469791.

[55] Lloyd Trefethen. Multivariate polynomial approximation in the hypercube. *Proceedings of the American Mathematical Society*, 145(11):4837–4844, 2017.

[56] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94: 103–114, 2017.

[57] Luca Dieci and Timo Eirola. Positive definiteness in the numerical solution of Riccati differential equations. *Numerische Mathematik*, 67(3):303–313, April 1994. ISSN 0945-3245. doi: 10.1007/s002110050030. URL https://doi.org/10.1007/s002110050030.

[58] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, March 1960. ISSN 0021-9223. doi: 10.1115/1.3662552. URL https://doi.org/10.1115/1.3662552.

[59] Hassan K. Khalil. *Nonlinear Systems, 3rd Edition*. Prentice Hall, 2002. URL https://www.pearson.com/content/one-dot-com/one-dot-com/us/en/higher-education/program.html.

[60] Multicopter PID Tuning Guide | PX4 User Guide, 12 2021. URL https://docs.px4.io/master/en/config_mc/pid_tuning_guide_multicopter.html.

ACKNOWLEDGEMENTS

Acknowledgements: A.A. is also affiliated with NVIDIA Corporation, and Y.Y. is also with associated Argo AI. K.A. is currently affiliated with Purdue University. We thank J. Burdick and J.-J. E. Slotine for their helpful discussions. We thank M. Anderson for help with configuring the quadrotor platform, and M. Anderson and P. Spieler for help with hardware troubleshooting. We also thank N. Badillo and L. Pabon Madrid for help in experiments.

Funding: 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). **Author contributions:** (1) S.-J.C. and Y.Y. directed the research activities. (2) G.S. and M.O'C. designed and implemented the meta-learning algorithm under the guidance of Y.Y., K.A., A.A., S.-J.C., while the last-layer adaptation idea was started with a discussion by G.S., M.O'C., X.S., and S.-J.C. (3) M.O'C., G.S. designed and implemented the adaptive control algorithm with inputs from S.-J.C., X.S.

(4) M.O'C., G.S. performed experiments and evaluated the results. (5) M.O'C. conducted the theoretical analysis of the meta-learning based adaptive controller with input from S.-J.C., G.S., X.S. (6) G.S. analyzed the learning algorithm with feedback from Y.Y., K.A., A.A., and S.-J.C. (7) G.S., M.O'C. made all the figures and videos with input from all the others. (8) All authors prepared the manuscript. **Competing interests:** The authors declare that they have no competing interests. **Data and materials availability:** All data needed to evaluate the conclusions in the article are present in the article or in the Supplementary Materials.

SUPPLEMENTARY MATERIALS

Section S1 Drone Configuration Details

Table S1 presents the configuration information of the custom built drone (fig. 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	$2.53\,\mathrm{kg}$	$1.47\,\mathrm{kg}$
Thrust-to-weight ratio	2.2	1.6
Rotor tilt angle	12° front, 10° rear	0°
Diameter	$85\,\mathrm{cm}$ wide, $75\,\mathrm{cm}$ long	$52\,\mathrm{cm}$ wide, $52\,\mathrm{cm}$ 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 S1: **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 authority 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 S2.

Section S2 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 $\bar{f}(x, w) : [-1, 1]^n \times [-1, 1]^m \to \mathbb{R}$ can be split into a w-invariant part $\bar{\phi}(x)$ and

	Neural-Fly	INDI [4]	Differentially flat linear drag [2]	Gaussian Process MPC [11]
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 S2: **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.

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 [55].

Lemma 2. *(Multivariate polynomial approximation in the hypercube) Let $\bar{f}(x,w) : [-1,1]^n \times [-1,1]^m \to \mathbb{R}$ 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 $[x,w] \in \mathbb{C}^{n+m}$ 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

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

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

Theorem 3. *$\bar{f}(x,w)$ is a function satisfying the assumptions in Lemma 2. For any $\epsilon > 0$, there exist $h \in \mathbb{Z}^+$, and two Chebyshev polynomials $\bar{\phi}(x) : [-1,1]^n \to \mathbb{R}^{1\times h}$ and $\bar{a}(w) : [-1,1]^m \to \mathbb{R}^{h\times 1}$ 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 $\sup_{[x,w]\in[-1,1]^{n+m}} \left\|\bar{f}(x,w) - \mathcal{C}_p(x,w)\right\| \leq \epsilon$. 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

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

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

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

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 3 can be generalized to vector-valued functions with bounded input space straightforwardly. Finally, since deep neural networks are universal approximators for polynomials [56], Theorem 3 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 $f(x,w)$.

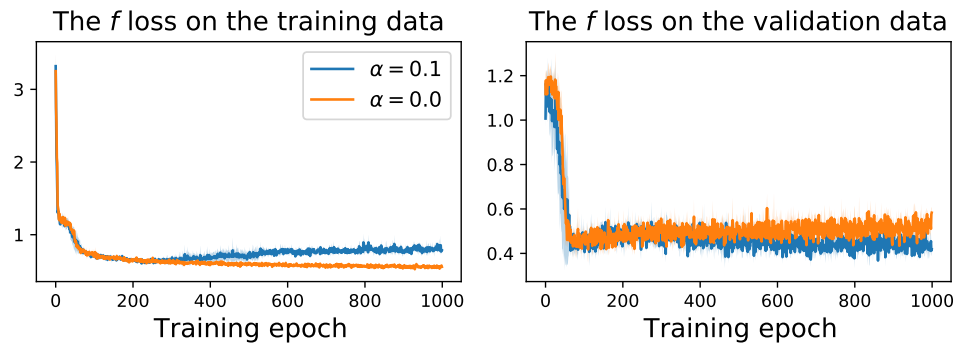

Figure S1: **Training and validation loss.** The evolution of the f loss on the training data and validation data in the training process, from three random seeds. Both mean (the solid line) and standard deviation (in the shaded area) are presented. Training with the adversarial regularization term ($\alpha = 0.1$) 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.

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
The degree of regularization α	0.1

Table S3: Hyperparameters used in DAIML (Algorithm 1).

Section S3 Hyperparameters for DAIML and the Interpretation

We implemented DAIML (Algorithm 1) using PyTorch, with hyperparameters reported in Table S3. We iteratively tuned these hyperparameters by trial and error. We notice that the behavior of the learning algorithm is not sensitive to most of parameters in Table S3. The training process is shown in fig. S1, 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 [52]. (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 Eq. (5). From fig. S1, 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. fig:training-tsne, $\alpha > 0$ disentangles the linear 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. S2 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.

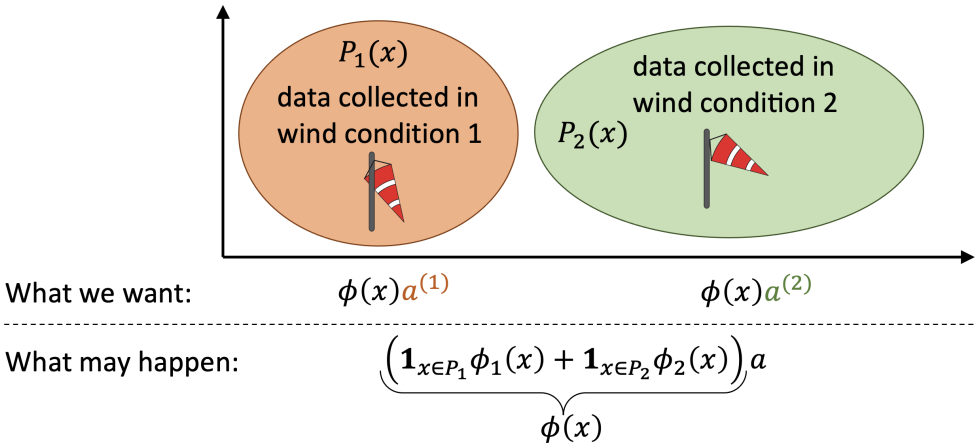

Figure S2: **Importance of domain-invariant representation.**

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 4 shows DAIML indeed makes the latent space much more disentangled.

Section S4 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 covaraince 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 [57] 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 Eq. (15) and (16) both $\hat{\mathbf{a}}$ and P are regularized. Then, in the update step in Eq. (18) and (19), P and $\hat{\mathbf{a}}$ are updated according to the gain in Eq. (17). This mirrors a discrete Kalman filter implementation [58] 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.

$$\hat{a}_{t_k^-} = \underbrace{(1 - \lambda \Delta t_k)}_{\text{damping}} \hat{a}_{t_{k-1}^+} \tag{15}$$

$$P_{t_k^-} = (1 - \lambda \Delta t_k)^2 P_{t_{k-1}^+} + Q \Delta t_k \tag{16}$$

$$K_{t_k} = P_{t_k^-} \phi_{t_k}^\top \left(\phi_{t_k} P_{t_k^-} \phi_{t_k}^\top + R \Delta t_k \right)^{-1} \tag{17}$$

$$\hat{a}_{t_k^+} = \hat{a}_{t_k^-} - \underbrace{K_{t_k} \left(\phi_{t_k} \hat{a}_{t_k^-} - y_{t_k} \right)}_{\text{prediction error adaptation}} - \underbrace{P_{t_k^-} \phi_{t_k}^\top s_{t_k}}_{\text{tracking error adaptation}} \tag{18}$$

$$P_{t_k^+} = \left(I - K_{t_k} \phi_{t_k} \right) P_{t_k^-} \left(I - K_{t_k} \phi_{t_k} \right)^\top + K_{t_k} R \Delta t_k K_{t_k}^\top \tag{19}$$

Section S5 Stability and Robustness Formal Guarantees and Proof

We divide the proof of Eq. (12) into two steps. First, in Theorem 4, 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 5 we show that when s is exponentially bounded, $\tilde{q}$ is also exponentially bounded. Combining the exponential bound from Theorem 4 and the ultimate bound from Corollary 5 proves Theorem 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 Eq. (1), control law Eq. (7), the composite velocity error definition Eq. (10), and the parameter estimation error $\tilde{a} = \hat{a} - a$, we find

$$M\dot{s} + (C + K)s = -\phi \tilde{a} + d \tag{20}$$

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

$$\lim_{t \to \infty} \|s\| \leq \frac{\sup_t \|d - \phi \tilde{a}\| \lambda_{\max}(M)}{\lambda_{\min}(K) \lambda_{\min}(M)} \tag{21}$$

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 [13]. 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 $\|\bar{\epsilon}\| \leq \|\epsilon\| + \|d\|$ by the triangle inequality. Using the above closed loop dynamics Eq. (20), the parameter estimation error $\tilde{a}$, and the adaptation law Eq. (8) and (9), the combined velocity and parameter-error closed-loop dynamics are given by

$$\begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} \dot{s} \\ \dot{\tilde{a}} \end{bmatrix} + \begin{bmatrix} C+K & \phi \\ -\phi^T & \phi^\top R^{-1}\phi + \lambda P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} = \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix} \tag{22}$$

$$\frac{d}{dt}\left(P^{-1}\right) = -P^{-1}\dot{P}P^{-1} = P^{-1}\left(2\lambda P - Q + P\phi^\top R^{-1}\phi P\right)P^{-1} \tag{23}$$

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 [57] show the slightly weaker result that that P is positive definite and finite when ϕ is bounded under the looser assumption $Q \succeq 0$. Following the proof from [57] 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. *Given dynamics that evolve according to Eq.* (22) *and* (23)*, 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 Eq.* (24) *with rate α.*

$$\lim_{t\to\infty} \left\| \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \right\| \leq \frac{1}{\alpha \lambda_{\min}(\mathcal{M})} \left(\sup_t \|d\| + \sup_t(\|\phi^\top R^{-1}\bar{\epsilon}\|) + \lambda_{\max}(P^{-1}) \sup_t(\|\lambda a + \dot{a}\|) \right) \tag{24}$$

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 5 and Eq. (24)*, the bound in Eq.* (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 Eq.* (24) *small.*

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

$$\mathcal{V} = \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \tag{25}$$

This Lyapunov function has the derivative

$$\dot{\mathcal{V}} = 2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} \dot{s} \\ \dot{\tilde{a}} \end{bmatrix} + \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} \dot{M} & 0 \\ 0 & \frac{d}{dt}\left(P^{-1}\right) \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \tag{26}$$

$$= -2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} C+K & \phi \\ -\phi^T & \phi^\top R^{-1}\phi + \lambda P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} + 2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix} + \tag{27}$$

$$\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} \dot{M} & 0 \\ 0 & \frac{d}{dt}\left(P^{-1}\right) \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \tag{28}$$

$$= -2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} K & \phi \\ -\phi^T & \phi^\top R^{-1}\phi + \lambda P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} + 2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix} \tag{29}$$

$$+ \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} 0 & 0 \\ 0 & 2\lambda P^{-1} - P^{-1}QP^{-1} + \phi^\top R^{-1}\phi \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \tag{30}$$

$$= -\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} 2K & 0 \\ 0 & \phi^\top R^{-1}\phi + P^{-1}QP^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} + 2\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix} \tag{31}$$

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

$$-\begin{bmatrix} 2K & 0 \\ 0 & \phi^\top R^{-1}\phi + P^{-1}QP^{-1} \end{bmatrix} \preceq -2\alpha \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \tag{32}$$

for all t.

Define an upper bound for the disturbance term D as

$$D = \sup_t \left\| \begin{bmatrix} d \\ \phi^\top R^{-1}\bar{\epsilon} - P^{-1}\lambda a - P^{-1}\dot{a} \end{bmatrix} \right\| \tag{33}$$

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

$$\mathcal{M} = \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \tag{34}$$

By Eq. (32), the Cauchy-Schwartz inequality, and the definition of the minimum eigenvalue, we have the following inequality for $\dot{\mathcal{V}}$:

$$\dot{\mathcal{V}} \leq -2\alpha\mathcal{V} + 2\sqrt{\frac{\mathcal{V}}{\lambda_{\min}(\mathcal{M})}}D \tag{35}$$

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

$$2\dot{\mathcal{W}}\mathcal{W} \leq -2\alpha\mathcal{W}^2 + \frac{2D\mathcal{W}}{\sqrt{\lambda_{\min}(\mathcal{M})}} \tag{36}$$

$$\dot{\mathcal{W}} \leq -\alpha\mathcal{W} + \frac{D}{\sqrt{\lambda_{\min}(\mathcal{M})}} \tag{37}$$

By the Comparison Lemma [59],

$$\sqrt{\mathcal{V}} = \mathcal{W} \leq \mathrm{e}^{-\alpha t}\left(\mathcal{W}(0) - \frac{D}{\alpha\sqrt{\lambda_{\min}(\mathcal{M})}}\right) + \frac{D}{\alpha\sqrt{\lambda_{\min}(\mathcal{M})}} \tag{38}$$

and the stacked state exponentially converges to the ball

$$\lim_{t\to\infty}\left\|\begin{bmatrix}s\\ \tilde{a}\end{bmatrix}\right\| \leq \frac{D}{\alpha\lambda_{\min}(\mathcal{M})} \tag{39}$$

This completes the proof. □

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

Corollary 5. *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*

$$\|\tilde{q}\| \leq \mathrm{e}^{-\lambda_{\min}(\Lambda)t}\|\tilde{q}(0)\| + \int_0^t \mathrm{e}^{-\lambda_{\min}(\Lambda)(t-\tau)}A\mathrm{e}^{-\alpha\tau}\mathrm{d}\tau + \int_0^t \mathrm{e}^{-\lambda_{\min}(\Lambda)(t-\tau)}\frac{B}{\alpha}\mathrm{d}\tau \tag{40}$$

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

$$\lim_{t\to\infty}\|\tilde{q}\| \leq \frac{B}{\alpha\lambda_{\min}(\Lambda)} \tag{41}$$

Proof. From the Comparison Lemma [59], we can easily show Eq. (40). This can be further reduced as follows.

$$\|\tilde{q}\| \leq \mathrm{e}^{-\lambda_{\min}(\Lambda)t}\|\tilde{q}(0)\| + A\mathrm{e}^{-\lambda_{\min}(\Lambda)t}\int_0^t \mathrm{e}^{(\lambda_{\min}(\Lambda)-\alpha)\tau}\mathrm{d}\tau + \int_0^t \mathrm{e}^{-\lambda_{\min}(\Lambda)(t-\tau)}\frac{B}{\alpha}\mathrm{d}\tau \tag{42}$$

$$\leq \mathrm{e}^{-\lambda_{\min}(\Lambda)t}\|\tilde{q}(0)\| + A\frac{\mathrm{e}^{-\alpha t} - \mathrm{e}^{-\lambda_{\min}(\Lambda)t}}{\lambda_{\min}(\Lambda) - \alpha} + \frac{B\left(1 - \mathrm{e}^{-\lambda_{\min}(\Lambda)t}\right)}{\alpha\lambda_{\min}(\Lambda)} \tag{43}$$

Taking the limit, we arrive at Eq. (41)

□

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 6. *If $\phi \equiv 0$, then the bound in Eq.* (24) *can be simplified to*

$$\lim_{t\to\infty}\left\|\begin{bmatrix}s\\ \tilde{a}\end{bmatrix}\right\| \leq \frac{\sup\|d\| + \lambda_{\max}(P^{-1})\sup(\|\lambda a + \dot{a}\|)}{\min\left(\lambda, \lambda_{\min}(K)/\lambda_{\max}(M)\right)\lambda_{\min}(\mathcal{M})} \tag{44}$$

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

$$\alpha = \min\left(\frac{1}{2}\lambda_{\min}(P^{-1}Q), \frac{\lambda_{\min}(K)}{\lambda_{\max}(M)}\right) \tag{45}$$

$\phi \equiv 0$ 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 Eq. (44).

□

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 7. *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 Eq.* (24) *can be simplified to*

$$\lim_{t\to\infty} \left\| \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \right\| \leq \frac{\left(\left(1 + r^{-1}\right) \sup_t \|f - a\| + \epsilon/r\right) \lambda_{\max}(M)}{\lambda_{\min}(K)\lambda_{\min}(\mathcal{M})} \tag{46}$$

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

$$\frac{dp}{dt} = q - p^2/r \tag{47}$$

where $P(t) = p(t)I$. This equation can be integrated to find that p exponentially converges to $p = \sqrt{qr}$. Then by Eq. (32), $\alpha \leq \sqrt{q/r}$ 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

$$\lim_{t\to\infty} \left\| \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \right\| \leq \frac{D\lambda_{\max}(M)}{\lambda_{\min}(K)\lambda_{\min}(\mathcal{M})} \tag{48}$$

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

$$D \leq \left(1 + r^{-1}\right) \sup_t \|f - a\| + \epsilon/r \tag{49}$$

□

Section S6 Gain Tuning

The attitude controller was tuned following the method in [60]. The gains for all of the position controllers tested were tuned on a step input of $1\,\mathrm{m}$ 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 $5\,\mathrm{Hz}$. 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.

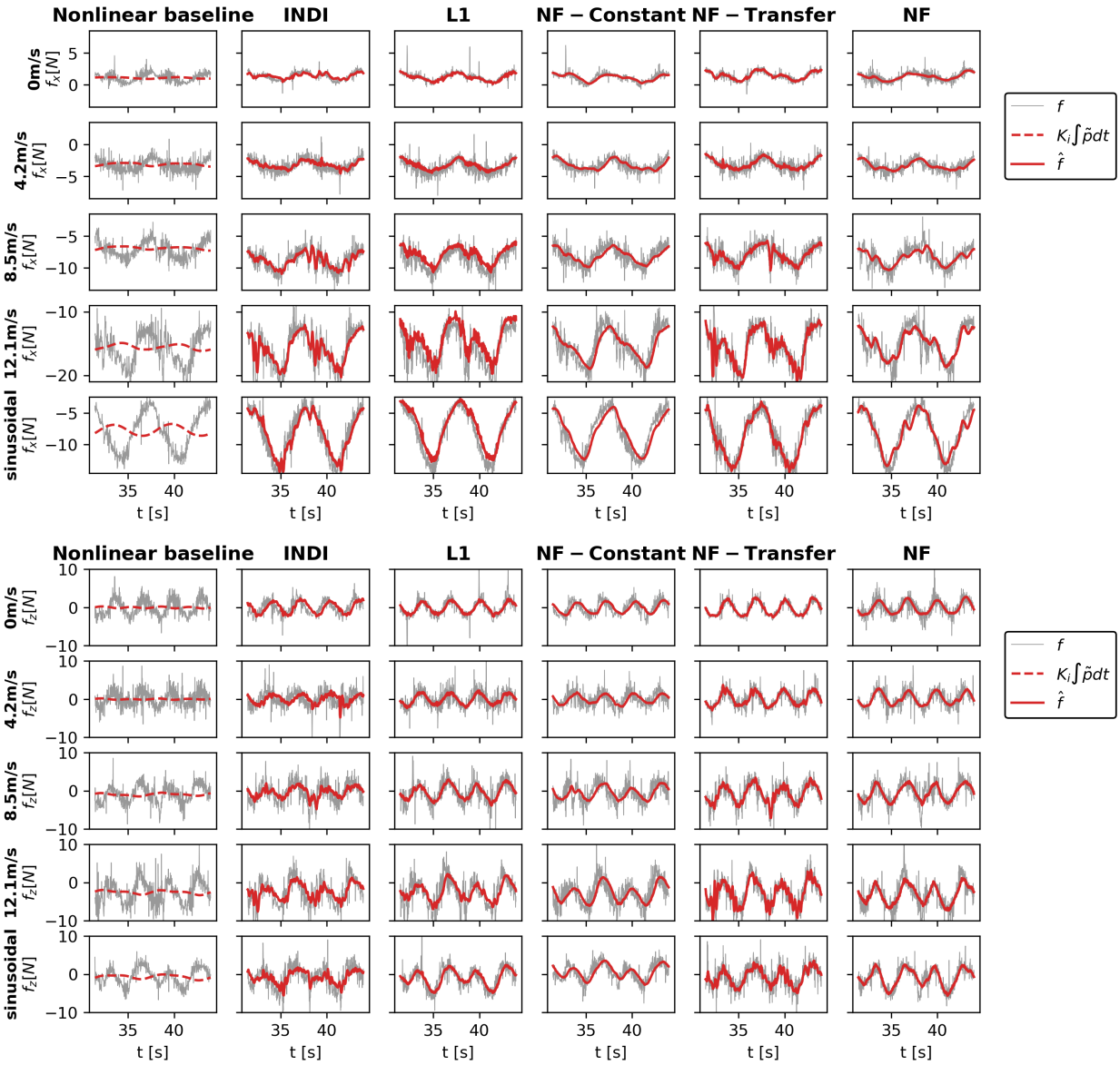

Figure S3: **Measured residual force versus adaptive control augmentation, $\hat{f}$.** Wind-effect x- and z-axis force prediction for different methods, $\hat{f}$ and $K_i \int \tilde{p}\mathrm{d}t$, compared with the online residual force measurement, f. The integral term in the nonlinear baseline 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.

Section S7 Force Prediction Performance

The section discusses fig. S3, 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. S3. Thus, the UAV trajcetory remains roughly centered on the desired trajectory for all wind conditions, as seen in fig. 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.

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. S3, we observe that each of these methods has a slight lag behind the measured residual force, in grey. 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. S3. 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.

Section S8 Localization Error Analysis

We estimate the root mean squared position localization precision to be about $1\,\mathrm{cm}$. 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 measurment 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. S4. 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 the OptiTrack 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 of the data from experiments in this paper, we see that the standard deviation is $1.0\,\mathrm{cm}$. Thus, we estimate that the localization precision has a standard deviation of about $1.0\,\mathrm{cm}$.

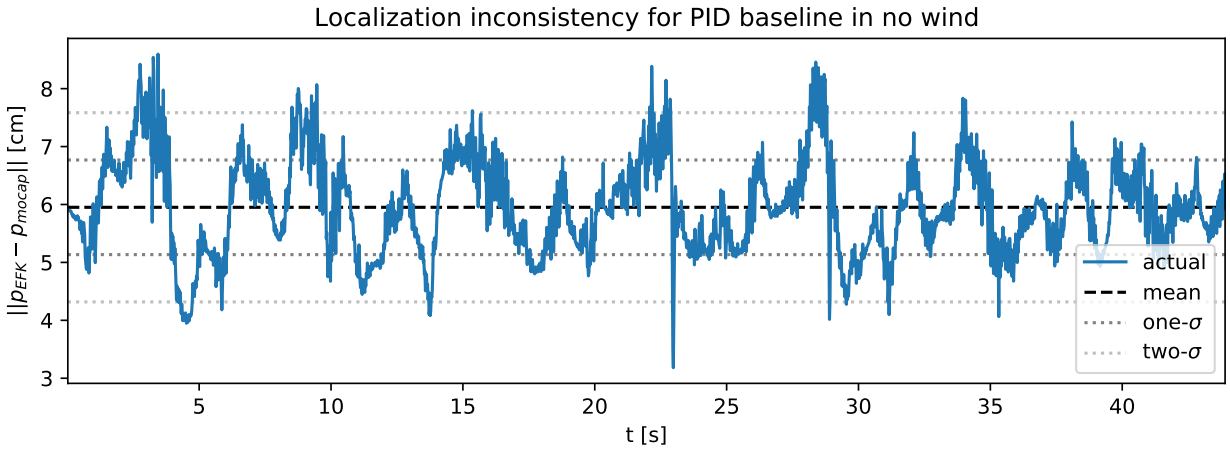

Figure S4: **Localization inconsistency** Typical difference between the OptiTrack motion capture position measurement, p_{mocap}, and the EKF position estimate, p_{EKF}, corrected for the Optitrack delay. The mean difference corresponds to a constant offset between the center of mass, which the EKF 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.