

AUTONOMOUS VEHICLES

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*

Copyright © 2022
The Authors, some
rights reserved;
exclusive licensee
American Association
for the Advancement
of Science. No claim
to original U.S.
Government Works

Executing safe and precise flight maneuvers in dynamic high-speed winds is important for the ongoing commoditization of uninhabited aerial vehicles (UAVs). However, because 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 pretrained 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 kilometers/hour (12.1 meters/second), 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. Last, our control design extrapolates to unseen wind conditions, is shown to be effective for outdoor flights with only onboard sensors, and can transfer across drones with minimal performance degradation.

INTRODUCTION

The commoditization of uninhabited aerial vehicles (UAVs) requires that the control of these vehicles becomes 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; and urban air mobility needs a flying car to follow a planned trajectory closely to avoid collision in the presence of strong unpredictable winds.

Unmodeled and often complex aerodynamics are among the most notable challenges to precise flight control. Flying in windy environments (as shown in Fig. 1) introduces even more complexity because of the unsteady aerodynamic interactions between the drone, the induced airflow, and the wind (see Fig. 1F 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 modeling is often computationally expensive and is limited to steady nondynamic wind conditions. Adaptive control addresses this problem by estimating linear parametric uncertainty in the dynamical model in real time to improve tracking performance. Recent state of the art in quadrotor flight control has used adaptive control methods that directly estimate the unknown aerodynamic force without assuming

the structure of the underlying physics, but relying on high-frequency and low-latency control (4–7). In parallel, there has been increased interest in data-driven modeling of aerodynamics [e.g., (8–11)]; however, existing approaches cannot effectively adapt in changing or unknown environments, such as time-varying wind conditions.

In this article, we present a data-driven approach called Neural-Fly, which is a deep learning-based trajectory tracking controller that learns to quickly adapt to rapidly 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 experiment 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 a total of 12 min of flight data in just six different wind conditions to train the DNN. DAIML incorporates several key features that 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.

Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA, USA.

*Corresponding author. Email: sjchung@caltech.edu

†These authors contributed equally to this work and are listed in alphabetical order.

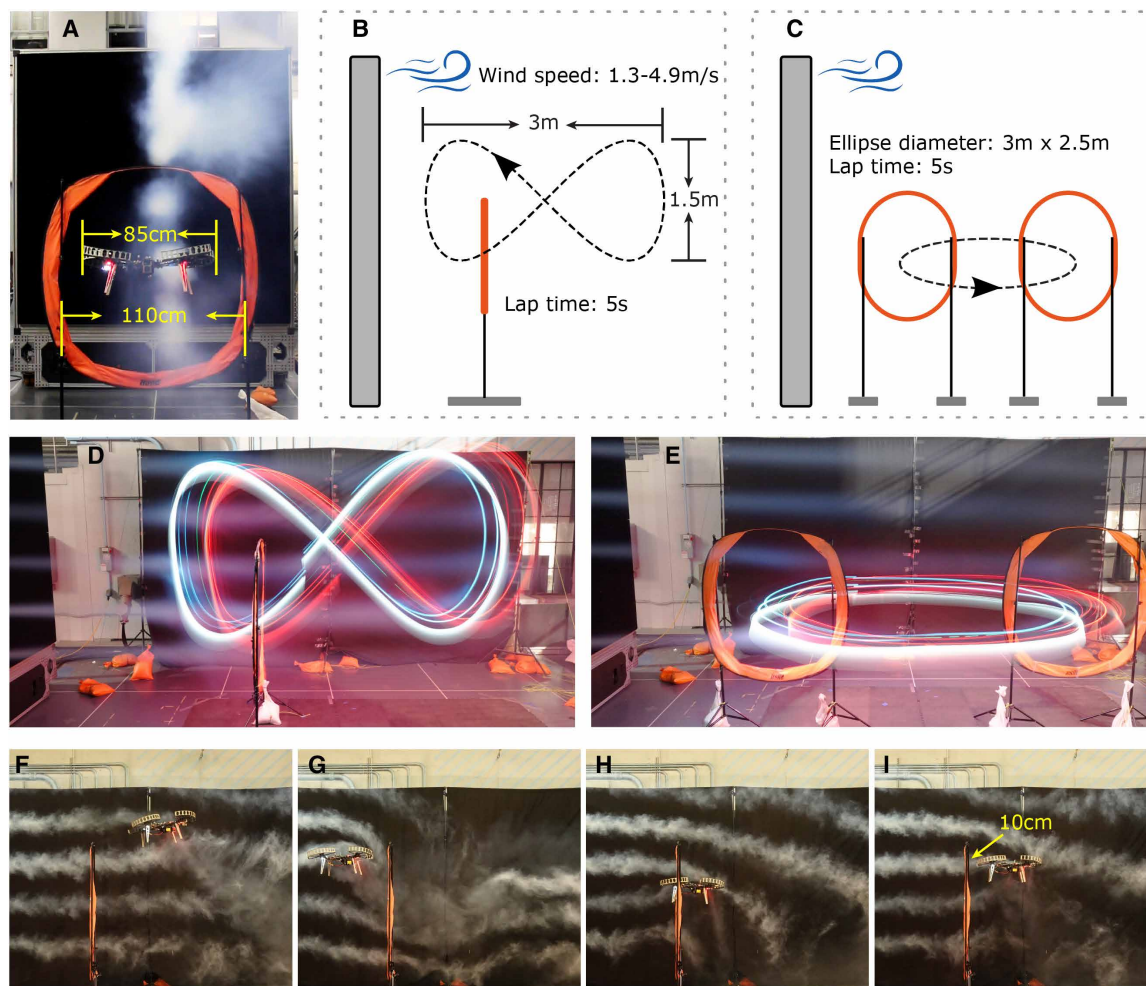


Fig. 1. Agile flight through narrow gates. (A) Caltech Real Weather Wind Tunnel system, the quadrotor UAV, and the gate. In our flight tests, the UAV follows an agile trajectory through narrow gates, which are slightly wider than the UAV itself, under challenging wind conditions. (B and C) Trajectories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

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

Using Neural-Fly, we report an average improvement of 66% over a nonlinear tracking controller, 42% over an \mathcal{L}_1 adaptive controller, and 35% over an incremental nonlinear dynamics inversion (INDI) controller. These results are all accomplished using standard quadrotor UAV hardware while running the PX4's default regulation attitude control. Our tracking performance is competitive even compared with related work without external wind disturbances

and with more complex hardware [for example, (4) requires a 10 times 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 nonlearning basis. NeuralFly-Transfer demonstrates that our method is robust to changes in vehicle configuration and model mismatch. NeuralFly-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 after 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 that separates the design of the position controller, attitude controller, and thrust mixer (allocation). Commonly used

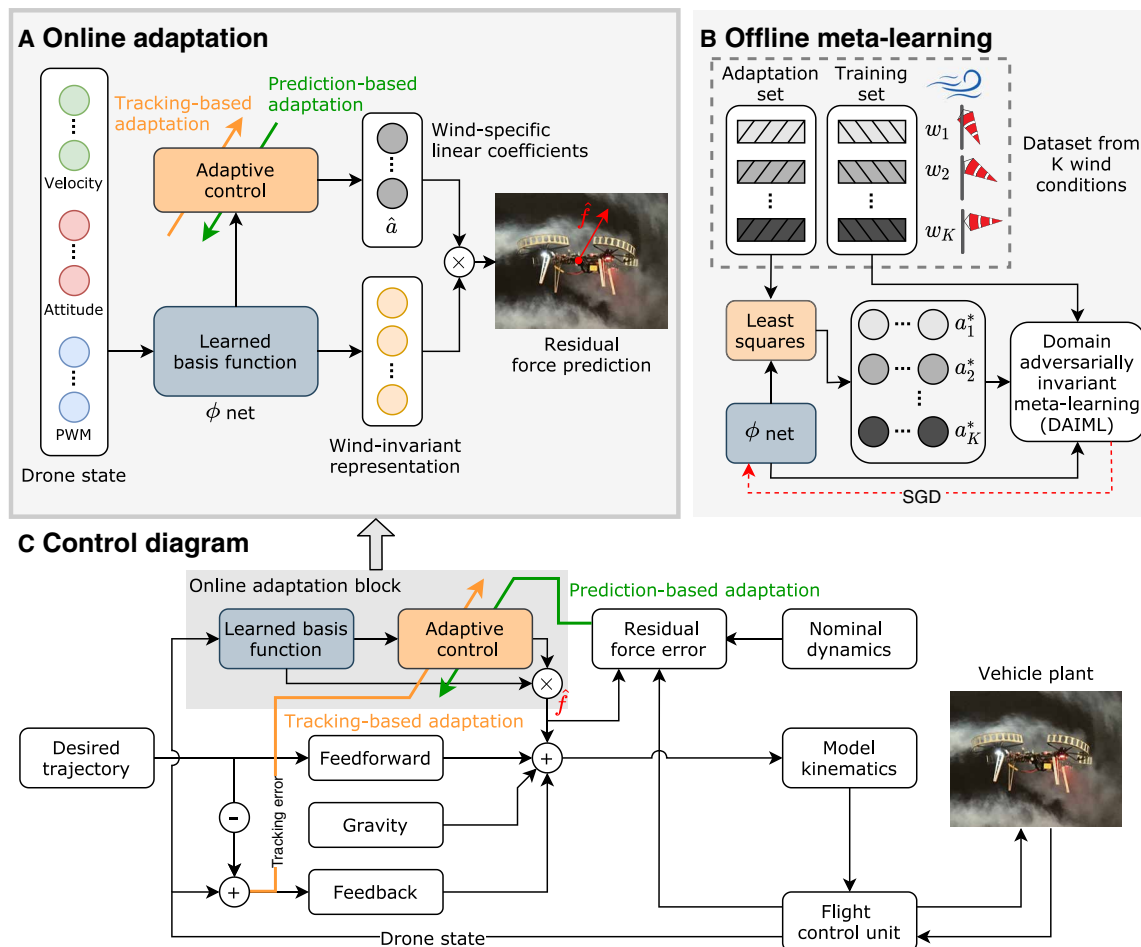


Fig. 2. Offline meta-learning and online adaptive control design. (A) The online adaptation block in our adaptive controller. Our controller leverages the meta-trained basis function ϕ , which is a wind-invariant representation of the aerodynamic effects, and uses composite adaptation (that is, including 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 $\{w_1, \dots, w_K\}$ and applied Algorithm 1 to train the ϕ net. (C) The diagram of our control method, where the gray part corresponds to (A). Interpreting the learned block as an aerodynamic force allows it to be incorporated into the feedback control easily.

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

Prior work on agile quadrotor control has achieved impressive results by considering aerodynamics (2, 4, 7, 11). However, those approaches require specialized onboard hardware (4), entail full custom flight control stacks (4, 7), or cannot adapt to external wind disturbances (2, 11). For example, state-of-the-art tracking performance has been demonstrated using INDI to estimate aerodynamic disturbance forces, with a root mean square tracking error of 6.6 cm

and drone ground speeds up to 12.9 m/s (4). However, Tai and Karaman (4) rely on high-frequency control updates (500 Hz) and direct motor speed feedback using optical encoders to rapidly estimate external disturbances. Both are challenging to deploy on standard systems. Hanover *et al.* (7) simplify the hardware setup, do not require optical motor speed sensors, and have demonstrated state-of-the-art tracking performance. However, Hanover *et al.* (7) rely 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. Torrente *et al.* (11) leverage an aerodynamic model learned offline represented as Gaussian processes. However, Torrente *et al.* (11) cannot adapt to unknown or changing wind conditions and provide no theoretical guarantees. Another recent work focuses on deriving simplified rotor-drag models that are differentially flat (2). However, the work of Faessler *et al.* (2) focuses on horizontal, x - y plane trajectories at ground speeds of 4 m/s without external wind, where the thrust is more constant than ours, achieves ~6-cm tracking error (2), uses an attitude controller running at 4000 Hz, and is not extensible to faster flights as pointed out in (11).

Relation between Neural-Fly and conventional adaptive control

Adaptive control theory has been extensively studied for online control and identification problems with parametric uncertainty, for example, unknown linear coefficients for mixing known basis functions (15–20). There are three common aspects of adaptive control that must be addressed carefully in any well-designed system and that 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 because of the coupling between the unknown underlying dynamics and the online adaptation. This coupling precludes naive combinations of online learning and control. For example, gradient-based parameter adaptation has well-known stability and robustness issues as discussed in (15).

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

Given a set of basis functions, naive adaptive control designs may cause instability and fragility in the closed-loop system, due to the nontrivial coupling between the adapted model and the system dynamics. In particular, asymptotically stable adaptive control cannot guarantee robustness against disturbances, and so exponential stability is desired. Even so, often existing adaptive control methods only guarantee exponential stability when the desired trajectory is persistently exciting, by which information about all 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 an MPC loop in (7)—achieve good results by abandoning complex basis functions. Instead, these methods directly estimate the aerodynamic residual force vector. The residual force is observable; thus, these methods bypass the challenge of designing good basis functions and the associated stability and persistent excitation issues. However, these methods suffer from lag in estimating the residual force and encounter the filter design performance trade of reduced lag versus amplified noise. Neural-Fly-Constant only uses Neural-Fly's composite adaptation law to estimate the residual force, and therefore, Neural-Fly-Constant also falls into this class of adaptive control structures. The results of this article demonstrate that the inherent estimation lag in these existing methods limits performance on agile trajectories and in strong wind conditions.

Neural-Fly solves the aforementioned issues of basis function design and adaptive control stability using newly developed methods for meta-learning and composite adaptation that can be seamlessly integrated together. NeuralFly 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 that are adapted online (19, 24–27). A recent work applies this idea to perform an impressive quadrotor flip (28). However, the existing neural network–based adaptive control work does not use multilayer DNNs and lacks a principled and efficient mechanism to pretrain the neural network before deployment. Instead of using shallow neural networks, recent trends in machine learning highly rely on DNNs due to their representation power (29). In this work, we leverage modern deep learning advances to pretrain a DNN that represents the underlying physics compactly and effectively.

Related work in multienvironment deep learning for robot control

Recently, researchers have been addressing the data and computation requirements for DNNs to help the field progress toward the fast online-learning paradigm. In turn, this progress has been enabling adaptable DNN-based control in dynamic environments. The most popular learning scheme in dynamic environments is meta-learning, or “learning to learn,” which aims to learn an efficient model from data across different tasks or environments (30, 31). 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 (32, 33), changing suspended payload for drones (34), and unknown operating conditions for wheeled robots (35).

In general, learning algorithms typically can be decomposed into two phases: offline learning and online adaptation. In the offline learning phase, the goal is to learn a model from data collected in different environments, such that the model contains shared knowledge or features across all environment, for example, learning aerodynamic features shared by all wind conditions. In the online adaptation phase, the goal is to adapt the offline-learned model, given limited online data from a new environment or a new task, for example, fine-tuning the aerodynamic features in a specific wind condition.

There are two ways that the offline-learned model can be adapted. In the first class, the adaptation phase adapts the whole neural network model, typically using one or more gradient descent steps (30, 32, 34, 36). However, because of the notoriously data-hungry and high-dimensional nature of neural networks, for real-world robots, it is still impossible to run such adaptation onboard as fast as the feedback control loop (e.g., ~100 Hz 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 (35, 37–39). The intuition is that different environments share a common representation (e.g., the wind-invariant representation in Fig. 2A), and the environment-specific part is in a low-dimensional space (e.g., the wind-specific linear weight in Fig. 2A), 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 (37), later followed by Richards *et al.* (38). However, the representation learned in (37) is ineffective, and the tracking performance in (37) is similar as the baselines; Richards *et al.* (38) focus on a planar and fully actuated rotorcraft simulation without experiment validation, and there is no stability or robustness analysis. Neural-Fly instead learns an effective representation using our meta-learning algorithm called DAIML, demonstrates state-of-the-art tracking performance on real drones, and achieves nontrivial stability and robustness guarantees.

Another popular deep-learning approach for control in dynamic environments is robust policy learning via domain randomization (40–42). 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 (40) 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.

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 that our methods reduce the average tracking error substantially. 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. Last, we show that our methods are also applicable in outdoor agile tracking tasks without external motion capture systems.

Experimental platform

All of our experiments were conducted at Caltech's Center for Autonomous Systems and Technologies. The experimental setup consisted of an OptiTrack motion capture system with 12 infrared cameras for localization streaming position measurements at 50 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 km/hour in its 3 m by 3 m by 5 m test section. For outdoor flight, the drone was 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. 1A. The quadrotor weighs 2.6 kg with a thrust-to-weight ratio of 2.2. The UAV is equipped with a Pixhawk flight controller running PX4, an open-source commonly used drone autopilot platform (13). The UAV incorporates a Raspberry Pi 4 onboard computer running a Linux operation system, which performs real-time computation and adaptive control and interfaces with the flight controller through Robot Operating System (ROS). State estimation is performed using the built-in PX4 Extended Kalman Filter (EKF), which fuses inertial measurement unit data with global position estimates from OptiTrack motion capture system (or the GPS module for outdoor flight tasks). The UAV platform features a wide-X configuration—measuring 85 cm in width, 75 cm in length, and 93 cm diagonally—and tilted motors for improved yaw authority. This general hardware setup is standard and similar to many quadrotors. We refer to the Supplementary Materials (section S1) for further configuration details.

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

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

Wind condition design

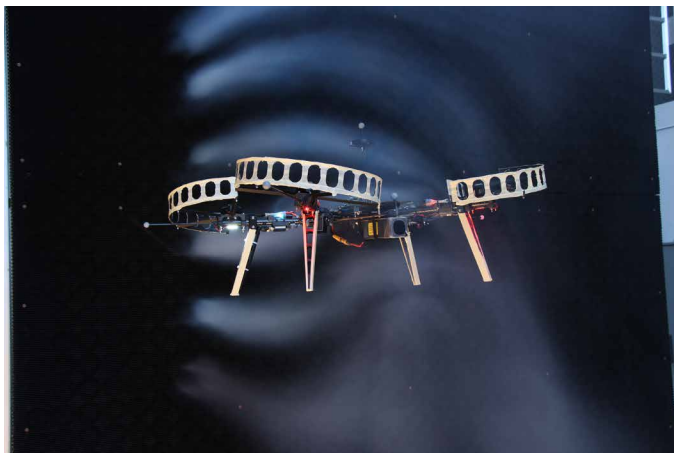
To generate dynamic and diverse wind conditions for the data collection and experiments, we leveraged the state-of-the-art Caltech Real Weather Wind Tunnel System (Fig. 1A). The wind tunnel is a 3 m by 3 m array of 1296 independently controllable fans capable of generating wind conditions up to 43.6 km/hour. 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 and 43.6 km/hour (12.1 m/s). The second type of wind follows a sinusoidal function in time, e.g., $30.6 + 8.6 \sin(t)$ km/hour. Note that the training data only cover constant wind speeds up to 6.1 m/s. To visualize the wind, we used five smoke generators to indicate the direction and intensity of the wind condition (see examples in Fig. 1 and Movie 1).

Offline learning and online adaptive control development

Data collection and meta-learning using DAIML

To learn an effective representation of the aerodynamic effects, we had a custom-built drone follow a randomized trajectory for 2 min each in six different static wind conditions, with speeds ranging from 0 to 22.0 km/hour. However, in experiments, we used wind speeds up to 43.6 km/hour (12.1 m/s) (e.g., Fig. 6). Data were

collected at 50 Hz with a total of 36,000 data points. Figure 3A shows the data collection process, and Fig. 3B shows the inputs and labels of the training data, under one wind condition of 13.3 km/hour (3.7 m/s). Figure 3C shows the distributions of input data (pitch) and label data (x component of the aerodynamic force) in different wind conditions. 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 was



Movie 1. Neural-Fly enables agile quadrotor flights through low-clearance gates.

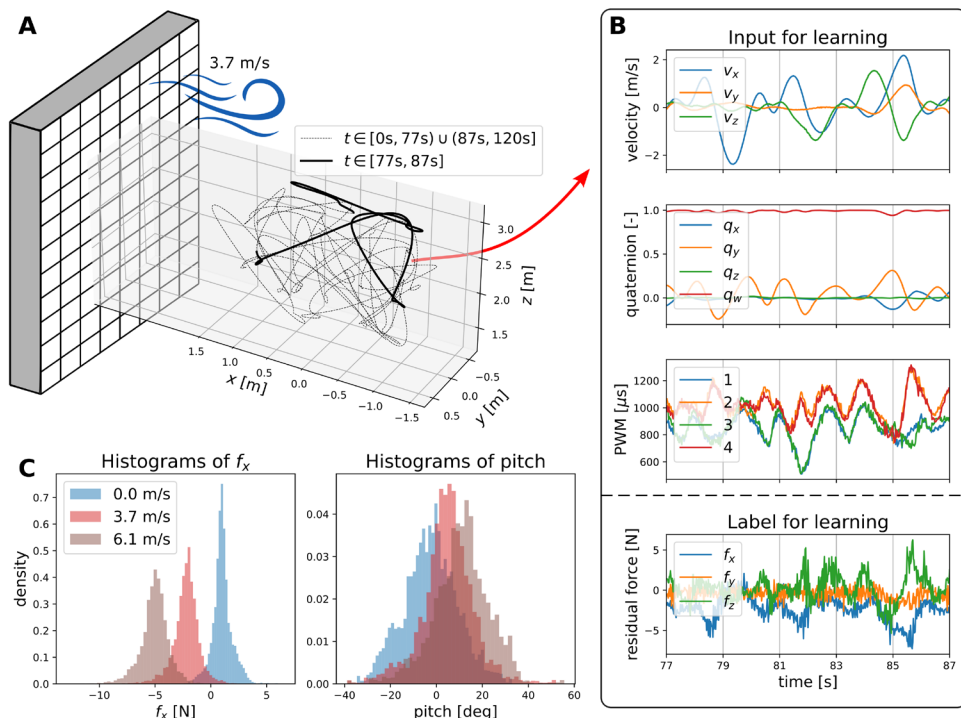


Fig. 3. Training data collection. (A) The xyz position along a 2-min randomized trajectory for data collection with a wind speed of 8.3 km/hour (3.7 m/s), in the Caltech Real Weather Wind Tunnel. (B) A typical 10-s trajectory of the inputs (velocity, attitude quaternion, and motor speed PWM command) 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. Left: Distributions of the x -component of the wind-effect force, f_x . This shows that the aerodynamic effect changes as the wind varies. 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.

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 applied the DAIML algorithm to learn two representations ϕ of the wind effects. The learning process was done offline on a normal desktop computer and is depicted in Fig. 2B. 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 Materials and Methods 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 Materials and Methods). Each of the controllers is implemented in the position control loop and outputs a force command. The force command is fed into a kinematics block to determine a corresponding attitude and thrust, similar to (14), which is sent to the PX4 flight controller. The three baselines include the following: globally exponentially stabilizing nonlinear tracking controller for quadrotor control (8, 43, 44), INDI linear acceleration control (4), and \mathcal{L}_1 adaptive control (5, 7). The primary difference between these baseline methods and Neural-Fly is how the controller compensates for the unmodeled

residual force (that is, each baseline method has the same control structure, in Fig. 2C, 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 and 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. NeuralFly-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

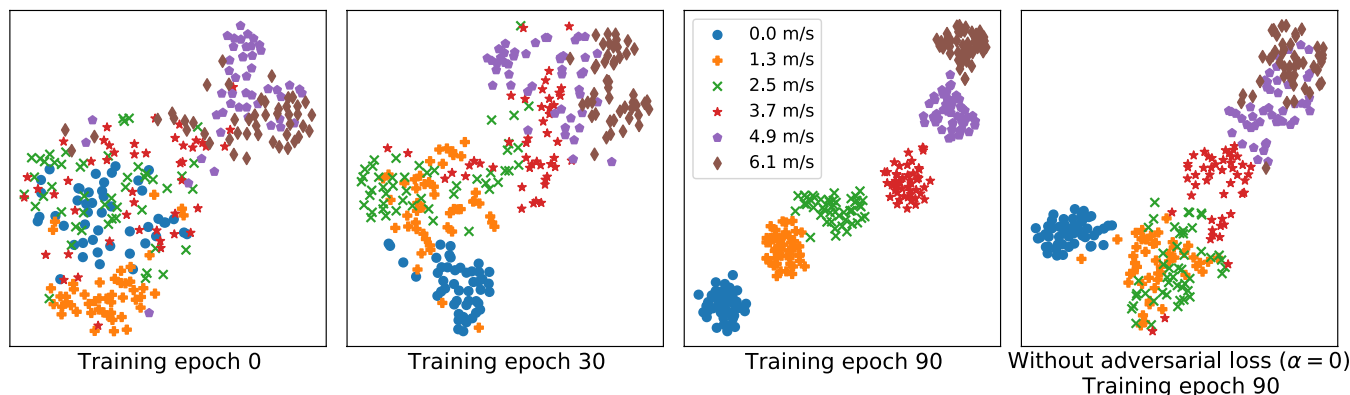
t-SNE plots of the linear coefficients (a^*) in the training process

Fig. 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 top right 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 with 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.

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 compared with data collection. Last, Neural-Fly-Constant demonstrates the benefit of using a better representation learned from the proposed meta-learning method DAIML. Note that Neural-Fly-Constant is a composite adaptation form of a Kalman filter disturbance observer, that is, a Kalman filter augmented with a tracking error update term.

Trajectory tracking performance

We quantitatively compare the performance of the aforementioned control methods when the UAV follows a 2.5-m-wide, 1.5-m-tall figure-8 trajectory with a lap time of 6.28 s under constant, uniform wind speeds of 0 km/hour, 15.1 km/hour (4.2 m/s), 30.6 km/hour (8.5 m/s), and 43.6 km/hour (12.1 m/s), and under time-varying wind speeds of $30.6 + 8.6 \sin(t)$ km/hour [$8.5 + 2.4 \sin(t)$ 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-s laps. The nonlinear baseline integral term compensates for the mean model error within the first lap. As the wind speed increases, the aerodynamic force variation becomes larger, and we notice a substantial performance degradation. INDI and \mathcal{L}_1 both improve over the nonlinear baseline, but INDI is more robust than \mathcal{L}_1 at high wind speeds. Neural-Fly-Constant outperforms INDI except during the two most challenging tasks: 43.6 km/hour 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 SD for the mean lap position error. In all cases, Neural-Fly and NeuralFly-Transfer outperform the state-of-the-art baseline methods, including the

30.6 km/hour, 43.6 km/hour, and sinusoidal wind speeds—all of which exceed the wind speed in the training data. All of these results present a trend: Adaptive control substantially outperforms the nonlinear baseline that relies on integral control, and learning markedly improves adaptive control.

Agile flight through narrow gates

Precise flight control in dynamic and strong wind conditions has many applications, such as rescue and search, delivery, and transportation. In this section, we present a challenging drone flight task in strong winds, where the drone must follow agile trajectories through narrow gates, which are only slightly wider than the drone. The overall result is depicted in Fig. 1 and Movie 1. As shown in Fig. 1A, the gates used in our experiments are 110 cm in width, which is only slightly wider than the drone (85 cm wide and 75 cm long). To visualize the trajectory using long-exposure photography, our drone was deployed with four main light-emitting diodes (LEDs) on its legs, where the two rear LEDs were red, and the front two were white. There were 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, and F to I) and Movie 1], the desired trajectory is a 3 m by 1.5 m figure-8 in the x - z plane with a lap time of 5 s. A gate is placed on the left bottom part of the trajectory. The minimum clearance is about 10 cm (see Fig. 1I), which requires that the controller precisely tracks the trajectory. The maximum speed and acceleration of the desired trajectory are 2.7 m/s and 5.0 m/s², respectively. The wind speed was 3.1 m/s. The second task (see Movie 1) is the same as the first one, except that it uses a more challenging, time-varying wind condition, $3.1 + 1.8 \sin(\frac{2\pi}{5}t)$ m/s. In the third task [see Fig. 1 (C and E) and Movie 1], the desired trajectory is a 3 m by 2.5 m ellipse in the x - y plane with a lap time of 5 s. We placed two gates on the left and right sides of the ellipse. As with the first task, the wind speed was 3.1 m/s.

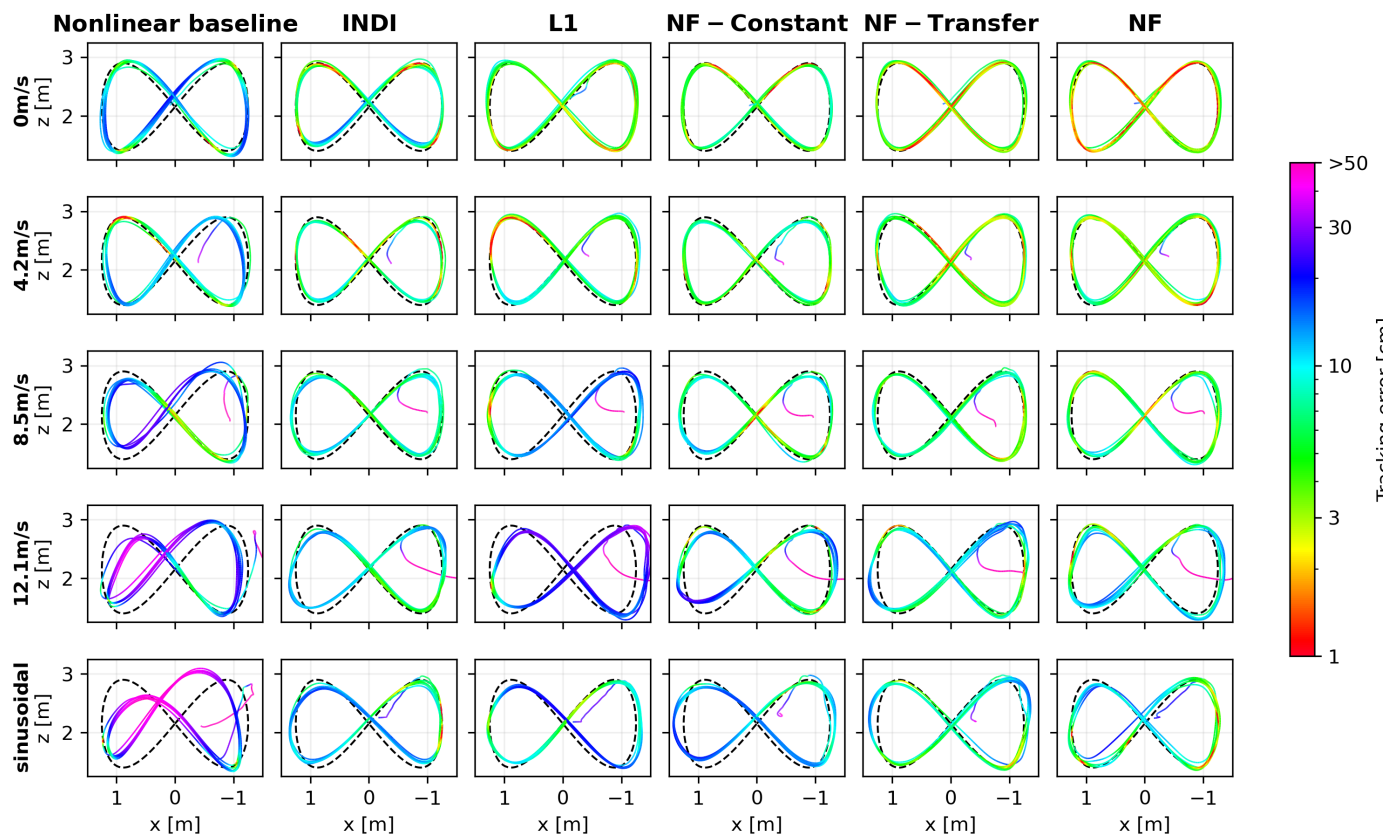


Fig. 5. Depiction of the trajectory tracking performance of each controller in several wind conditions. The baseline nonlinear controller can track the trajectory well; however, the performance substantially degrades at higher wind speeds. INDI, \mathcal{L}_1 , and Neural-Fly-Constant have similar performance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

Table 1. Tracking error statistics in centimeters 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

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 and E) shows two long-exposure photos with an exposure time of 5 s, which is the same as the lap time of the desired trajectory. We see that our method precisely tracked the desired trajectories and flew safely through the gates (see Movie 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, because the LEDs are located on the legs (see Fig. 1A). Figure 1 (F to I) shows four high-speed photos with a shutter speed of 1/200 s. These four photos captured the moment the drone passed through the gate in the first task and the complex interaction between the drone and the wind. We see that the aerodynamic effects are complex and nonstationary and depend on the UAV attitude, the relative velocity, and aerodynamic interactions between the propellers and the wind.

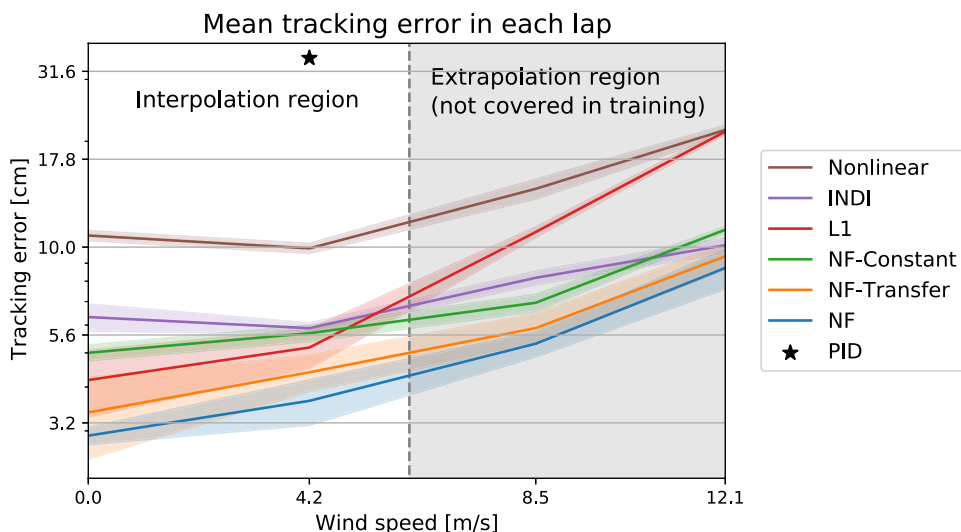


Fig. 6. Mean tracking errors of each lap in different wind conditions. This figure shows position tracking errors of different methods as wind speed increases. Solid lines show the mean error over six laps, and the shaded areas show SD of the mean error on each lap. The gray area indicates the extrapolation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

Outdoor experiments

We tested our algorithm outdoors in gentle breeze conditions (wind speeds measured up to 17 km/hour). An onboard GPS receiver provided position information to the EKF, giving lower precision state estimation and therefore less precise aerodynamic residual force estimation. After the same aforementioned figure-8 trajectory, the controller reached 7.5-cm mean tracking error, shown in Fig. 7.

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 was compared tracking an agile figure-8 in constant and time-varying wind speeds up to 43.6 km/hour (12.1 m/s). Furthermore, we observe a mean tracking error of 2.9 cm in 0 km/h wind, which is comparable with state-of-the-art tracking performance demonstrated on more aggressive racing drones (4, 7) despite several architectural limitations such as limited control rate in offboard mode, a larger, less maneuverable vehicle, and without direct motor speed measurements. All our experiments were conducted using the standard PX4 attitude controller, with Neural-Fly implemented in an onboard, low-cost, and “credit card-sized” Raspberry Pi 4 computer. Furthermore, Neural-Fly is robust to changes in vehicle configuration, as demonstrated by the similar performance of Neural-Fly-Transfer.

To understand the fundamental tracking-error limit, we estimate that the localization precision from the OptiTrack system is about 1 cm, which is a practical lower bound for the average tracking error in our system (see more details in the Supplementary Materials, 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 cm.

To achieve a tracking error of 1 cm, the remaining improvements should focus on reducing code execution time, communication delays, and attitude tracking delay. We measured the combined code execution time and communication delay to be at least 15 ms and often as much as 30 ms. A faster implementation (such as using C++ instead of Python) and streamlined communication layer (such as using ROS2’s real-time features) could allow us to achieve tracking errors on the order of the localization accuracy. Attitude tracking delay can be substantially reduced through the use of a nonlinear attitude controller [e.g., (44)]. Our method is also directly extensible to attitude control because attitude dynamics match the Euler-Lagrange dynamics used in our derivations. However, further work is needed to understand the interaction of the learned dynamics with the cascaded

control design when implementing a tracking attitude controller.

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

Challenges caused by unknown and time-varying wind conditions

In the real world, the wind conditions are not only unknown but also constantly changing, and the vehicle must continuously adapt. We designed the sinusoidal wind test to emulate unsteady or gusty wind conditions. Although our learned model was 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 repeatedly 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. 2A) and the wind condition (i.e., the wind-specific linear weight in Fig. 2A). This separation allows Neural-Fly to quickly adapt to the time-varying wind even as the UAV follows a dynamic trajectory, with an average tracking error below 8.7 cm in Table 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

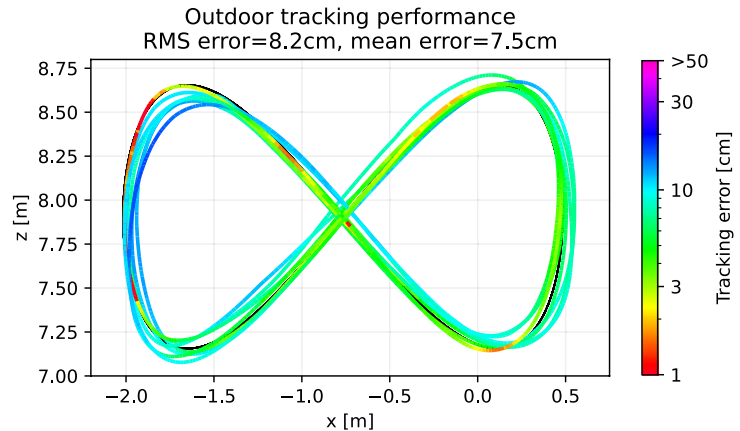
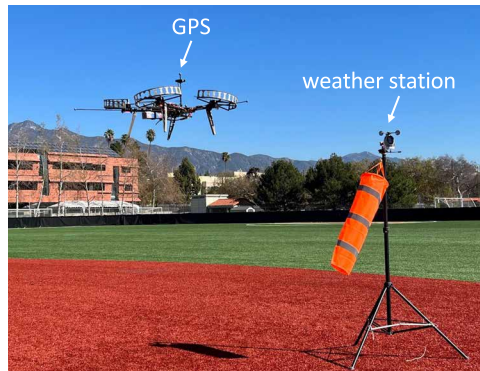


Fig. 7. Outdoor flight setup and performance. Left: In outdoor experiments, a GPS module is deployed for state estimation, and a weather station records wind profiles. The maximum wind speed during the test was around 17 km/hour (4.9 m/s). Right: Trajectory tracking performance of Neural-Fly.

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

Generalization to new trajectories and new aircraft

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 those of Foehn *et al.* (1) or learning-based planners (45, 46). In particular, for those planners that 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 shows that our approach is robust to changing vehicle design and that 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-Lagrange equation (see Materials and Methods), including many types of aircraft such as (21, 47).

MATERIALS AND METHODS

Overview

We consider a general robot dynamics model

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

where $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \in \mathbb{R}^n$ are the n dimensional position, velocity, and acceleration vectors; $M(\mathbf{q})$ is the symmetric, positive definite inertia matrix; $C(\mathbf{q}, \dot{\mathbf{q}})$ is the Coriolis matrix; $\mathbf{g}(\mathbf{q})$ is the gravitational force vector; and $\mathbf{u} \in \mathbb{R}^n$ is the control force. $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{w})$ incorporates unmodeled dynamics, and $\mathbf{w} \in \mathbb{R}^m$ is an unknown hidden state used to represent the underlying environmental conditions, which is potentially time-variant. Specifically, in this article, \mathbf{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

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

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 \mathbf{a} is a set of linear coefficients that is updated for each condition. In the Supplementary Material (section S2), we prove that the decomposition $\phi(\mathbf{q}, \dot{\mathbf{q}})\mathbf{a}(\mathbf{w})$ exists for any analytic function $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{w})$. In the offline meta-learning stage, we learn ϕ as a DNN using our metalearning 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 \mathbf{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. 2B and Fig. 2 (A and 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 min each in several different static wind conditions. Figure 3A illustrates one trajectory

under the wind condition 13.3 km/hour (3.7 m/s). The set of input-output pairs for the k th such trajectory is referred to as the k th subdataset, D_{w_k} , with the wind condition w_k . Our dataset consists of six different subdatasets with wind speeds from 0 to 22.0 km/hour (6.1 m/s), which are in the white interpolation region in Fig. 6.

The trajectory follows a polynomial spline between three waypoints: the current position and two randomly generated target positions. The spline is constrained to have zero velocity, acceleration, and jerk at the starting and ending waypoints. Once the end of one spline is reached, a new random spline is generated, and the process repeats for the duration of the training data flight. This process allows us to generate a large amount of data using a trajectory very different from the trajectories used to test our method, such as the figure-8 in Fig. 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 fifthorder 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. Last, this allows us to define the dataset, $\mathcal{D} = \{D_{w_1}, \dots, D_{w_K}\}$, where

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

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

The 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 $a(w)$ that allows $\phi(x)a(w)$ to approximate $f(x, w)$ well.

Formally, an optimal representation, ϕ , solves the following optimization problem

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

where $\phi(\cdot) : \mathbb{R}^{2n} \rightarrow \mathbb{R}^n \times h$ 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 DNN to represent ϕ . In the Supplementary Materials (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. 3C, 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 $\{f(x, w_1), f(x, w_2), \dots, f(x, w_K)\}$ is reflected via the distribution of x , rather than the wind condition $\{w_1, w_2, \dots, w_K\}$. In other words, the optimization in Eq. 4 may lead to overfitting and may not properly find a wind-invariant representation ϕ .

To solve the domain shift problem, inspired by the work of Ganin *et al.* (48), we propose the following adversarial optimization framework

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

where \mathbf{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, \mathbf{h} and ϕ play a zero-sum max-min game: The goal of \mathbf{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 \mathbf{h} harder (achieved by the inner min). In other words, \mathbf{h} is a learned regularizer to remove the environment information contained in ϕ . In our experiments, the output of \mathbf{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

$$\text{loss}(\mathbf{h}(\phi(x_k^{(i)})), k) = - \sum_{j=1}^K \delta_{kj} \log(\mathbf{h}(\phi(x_k^{(i)}))^T \mathbf{e}_j) \quad (6)$$

where $\delta_{kj} = 1$ if $k = j$ and $\delta_{kj} = 0$ otherwise, and \mathbf{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: $\mathcal{D} = \{D_{w_1}, \dots, D_{w_K}\}$

Initialize: Neural networks ϕ and \mathbf{h}

Result: Trained neural networks ϕ and \mathbf{h}

1 **repeat**

 lines 2–9 **until convergence**

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 $a^*(\phi) = \arg \min_a \sum_{i \in B^a} \|y_k^{(i)} - \phi(x_k^{(i)}) a\|^2$

5 **if** $\|a^*\| > \gamma$ **then**

6 $a^* \leftarrow \gamma \cdot \frac{a^*}{\|a^*\|}$

7 Train DNN ϕ using stochastic gradient descent (SGD) and spectral normalization with loss

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

8 if rand() $\leq \eta$ then

9 Train DNN \mathbf{h} using SGD with loss $\sum_{i \in B} \text{loss}(\mathbf{h}(\phi(\mathbf{x}_k^{(i)})), k)$

Design of the DAIML algorithm

Last, we solve the optimization problem in Eq. 5 by the proposed algorithm DAIML (described in Algorithm 1 and illustrated in Fig. 2B), which belongs to the category of gradient-based meta-learning (31), but with least squares as the adaptation step. DAIML contains three steps: (i) The adaptation step (lines 4 to 6) solves a least squares problem as a function of ϕ on the adaptation set B^a . (ii) The training step (line 7) updates the learned representation ϕ on the training set B , based on the optimal linear coefficient \mathbf{a}^* solved from the adaptation step. (iii) The regularization step (lines 8 and 9) updates the discriminator \mathbf{h} on the training set.

We emphasize the following important features of DAIML: (i) After the adaptation step, \mathbf{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 \mathbf{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 $\|\mathbf{a}^*\| \leq \gamma$, 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 \mathbf{h} and ϕ in an alternating manner. In each iteration, we first update ϕ (line 7) while fixing \mathbf{h} and then update \mathbf{h} (line 9) while fixing ϕ . However, the probability to update the discriminator \mathbf{h} in each iteration is $\eta \leq 1$ instead of 1, to improve the convergence of the algorithm (49).

We further motivate the algorithm design using Figs. 3 and 4. Figure 3 (A and B) shows the input and label from one wind condition, and Fig. 3C 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 (\mathbf{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 \mathbf{a}^* into 2-d. The distribution of \mathbf{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: The top right part of the t-SNE plot corresponds to a higher wind speed. These properties imply that the learned representation ϕ is shared by all wind conditions, and the linear weight t contains the wind-specific information. Last, 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 \mathbf{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 find a set of parameters \mathbf{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 on a Kalman filter estimator. This formulation results in automatic gain tuning for the update law, which allows the controller to quickly estimate parameters with large uncertainty. We further boost this estimator into a composite adaptation law; that is, the parameter update depends both on the prediction error in the dynamics model and on the tracking error, as illustrated in Fig. 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

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

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

$$\dot{P} = -2\lambda P + Q - P\phi^T R^{-1} \phi P \quad (9)$$

where \mathbf{u}_{NF} is the control law; $\hat{\mathbf{a}}$ is the online linear-parameter update; P is a covariance-like matrix used for automatic gain tuning; $\mathbf{s} = \dot{\mathbf{q}} + \Lambda \tilde{\mathbf{q}}$ is the composite tracking error; $\dot{\mathbf{q}}_r$ is the reference velocity, \mathbf{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 law \mathbf{u}_{NL} . Then, we discuss our control law \mathbf{u}_{NF} in detail. Note that \mathbf{u}_{NF} not only depends on the desired trajectory but also requires the learned representation ϕ and the linear parameter $\hat{\mathbf{a}}$ (an estimation of \mathbf{a}). The composite adaptation algorithm for $\hat{\mathbf{a}}$ 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 $\mathbf{d}(t)$ as the representation error: $\mathbf{f} = \phi \cdot \mathbf{a} + \mathbf{d}(t)$, and our theory shows that the robot tracking error exponentially converges to an error ball whose size is proportional to $\|\mathbf{d}(t) + \epsilon\|$ (i.e., the learning error and measurement noise) and $\|\dot{\mathbf{a}}\|$ (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 \mathbf{s} and the reference velocity $\dot{\mathbf{q}}_r$ are defined such that

$$\mathbf{s} = \dot{\mathbf{q}} - \dot{\mathbf{q}}_r = \dot{\tilde{\mathbf{q}}} + \Lambda \tilde{\mathbf{q}} \quad (10)$$

where $\tilde{\mathbf{q}} = \mathbf{q} - \mathbf{q}_d$ is the position tracking error and Λ is a positive definite gain matrix. Note that when \mathbf{s} exponentially converges to

an error ball around 0, \mathbf{q} will exponentially converge to a proportionate error ball around the desired trajectory $\mathbf{q}_d(t)$ (see section S5). Formulating our control law in terms of the composite velocity errors simplifies the analysis and gain tuning without loss of rigor.

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

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

where K and K_I are positive definite control gain matrices. Note that a standard PID controller typically only includes the PID feedback terms, 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., $M(\mathbf{q})$ and $C(\mathbf{q}, \dot{\mathbf{q}})$ are nonconstant in attitude control]. However, this control law only compensates for changing wind conditions and unmodeled dynamics through an integral term, which is slow to react to changes in the unmodeled dynamics and disturbance forces.

Our method improves the controller by predicting the unmodeled dynamics and disturbance forces, and in Table 1, we see a substantial improvement gained by using our learning method. Given the learned representation of the residual dynamics, $\phi(\mathbf{q}, \dot{\mathbf{q}})$, and the parameter estimate $\hat{\mathbf{a}}$, we replace the integral term with the learned force term, $\hat{\mathbf{f}} = \phi\hat{\mathbf{a}}$, 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 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 Eqs. 8 and 9, where \mathbf{y} is a noisy measurement of \mathbf{f} , λ is a damping gain, P is a covariance matrix that 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 (50). 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 \mathbf{d} and measurement noise ϵ . 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 the 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 when a new mode in the learned model is excited.

Stability and robustness guarantees

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

Theorem 1. *If we assume that the desired trajectory has bounded derivative and the system evolves according to the dynamics in Eq. 1, the control law (Eq. 7), and the adaptation law (Eqs. 8 and 9), then the position tracking error exponentially converges to the ball*

$$\lim_{t \rightarrow \infty} \|\tilde{\mathbf{q}}\| \leq \sup_t [C_1 \|\mathbf{d}(t)\| + C_2 \|\epsilon(t)\| + C_3 (\lambda \|\mathbf{a}(t)\| + \|\dot{\mathbf{a}}(t)\|)] \quad (12)$$

where C_1 , C_2 , and C_3 are three bounded constants depending on ϕ , R , Q , K , Λ , M , and λ .

Implementation details

Quadrotor dynamics

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

$$\dot{\mathbf{p}} = \mathbf{v}, m\dot{\mathbf{v}} = \mathbf{mg} + R\mathbf{f}_T + \mathbf{f} \quad (13a)$$

$$\dot{R} = RS(\boldsymbol{\omega}), \quad J\dot{\boldsymbol{\omega}} = J\boldsymbol{\omega} \times \boldsymbol{\omega} + \boldsymbol{\tau}_T \quad (13b)$$

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

We cast the position dynamics in Eq. 13a into the form of Eq. 1, by taking $M(\mathbf{q}) = mI$, $C(\mathbf{q}, \dot{\mathbf{q}}) \equiv 0$, and $\mathbf{u} = R\mathbf{f}_T$. Note that the quadrotor attitude dynamics (Eq. 13b) is also a special case of Eq. 1 (15, 51), 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 \mathbf{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 \mathbf{v} , the aerodynamic effects also depend on the drone attitude and the rotor rotation speed. To that end, the input state \mathbf{x} to the DNN ϕ is an

11-d vector, consisting of the drone velocity (3-d), the drone attitude represented as a quaternion (4-d), and the rotor speed commands as a pulse width modulation (PWM) signal (4-d) (see Figs. 2 and 3). The DNN ϕ has four fully connected hidden layers, with an architecture $11 \rightarrow 50 \rightarrow 60 \rightarrow 50 \rightarrow 4$ and rectified linear unit activation. We found that the three components of the wind-effect force, f_x , f_y , and f_z , are highly correlated and share common features, so we use ϕ as the basis function for all the components. Therefore, the wind-effect force \mathbf{f} is approximated by

$$\mathbf{f} \approx \begin{bmatrix} \phi(\mathbf{x}) & 0 & 0 \\ 0 & \phi(\mathbf{x}) & 0 \\ 0 & 0 & \phi(\mathbf{x}) \end{bmatrix} \begin{bmatrix} \mathbf{a}_x \\ \mathbf{a}_y \\ \mathbf{a}_z \end{bmatrix} \quad (14)$$

where $\mathbf{a}_x, \mathbf{a}_y, \mathbf{a}_z \in \mathbb{R}^4$ 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 Materials 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 $\mathbf{u} = \mathbf{R}\mathbf{f}_r$, which makes the controller law (e.g., Eq. 7) nonaffine in \mathbf{u} . We solve this issue by using the PWM from the last time step as an input to ϕ , to compute the desired force \mathbf{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 augmented the nonlinear baseline control with $\hat{\mathbf{f}} = -\mathbf{u}_{\mathcal{L}_1}$.

SUPPLEMENTARY MATERIALS

www.science.org/doi/10.1126/scirobotics.abm6597

Sections S1 to S8

Figs. S1 to S4

Tables S1 to S3

References (52–57)

REFERENCES AND NOTES

- P. Foehn, A. Romero, D. Scaramuzza, Time-optimal planning for quadrotor waypoint flight. *Sci. Robot.* **6**, abh1221 (2021).
- M. Faessler, A. Franchi, D. Scaramuzza, Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories. *IEEE Robot. Autom. Lett.* **3**, 620–626 (2018).
- P. Ventura Diaz, S. Yoon, High-fidelity computational aerodynamics of multi-rotor unmanned aerial vehicles, in *2018 AIAA Aerospace Sciences Meeting* (2018), p. 1266.
- E. Tai, S. Karaman, Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness. *IEEE Trans. Control Syst. Technol.* **29**, 1203–1218 (2021).
- S. Mallikarjunan, B. Nesbitt, E. Kharisov, E. Xargay, N. Hovakimyan, C. Cao, L1 adaptive controller for attitude control of multirotors, in *A/AA Guidance, Navigation, and Control Conference* (American Institute of Aeronautics and Astronautics, 2012).
- J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, E. A. Theodorou, LI-adaptive MPPI architecture for robust and agile control of multirotors, in *Proceedings of the 2020 IEEE/RSI International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 7661–7666. ISSN: 2153-0866.
- D. Hanover, P. Foehn, S. Sun, E. Kaufmann, D. Scaramuzza, Performance, precision, and payloads: Adaptive nonlinear MPC for quadrotors. *IEEE Robot. Autom. Lett.* **7**, 690–697 (2021).
- G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, S.-J. Chung, Neural lander: Stable drone landing control using learned dynamics, in *Proceedings of the 2019 International Conference on Robotics and Automation (ICRA)* (IEEE, 2019), pp. 9784–9790.
- G. Shi, W. Hönig, Y. Yue, S.-J. Chung, Neural-swarm: Decentralized close-proximity multirotor control using learned interactions, in *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2020), pp. 3241–3247.
- G. Shi, W. Hnig, X. Shi, Y. Yue, S.-J. Chung, Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions. *IEEE Trans. Robot.* **38**, 1063–1079 (2022).
- G. Torrente, E. Kaufmann, P. Föhn, D. Scaramuzza, Data-driven mpc for quadrotors. *IEEE Robot. Autom. Lett.* **6**, 3769–3776 (2021).
- P. L. Bartlett, D. J. Foster, M. J. Telgarsky, Spectrally-normalized margin bounds for neural networks. *Adv. Neural Inform. Process. Syst.* **30**, 6240 (2017).
- L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, M. Pollefeys, PIXHAWK—A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots* **33**, 21–39 (2012).
- D. Mellinger, V. Kumar, Minimum snap trajectory generation and control for quadrotors, in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation* (IEEE, 2011), pp. 2520–2525.
- J.-J. E. Slotine, W. Li, *Applied Nonlinear Control* (Prentice Hall, 1991).
- P. A. Ioannou, J. Sun, *Robust Adaptive Control* (Prentice-Hall Upper Saddle River, 1996), vol. 1.
- M. Krstic, P. V. Kokotovic, I. Kanellakopoulos, *Nonlinear and Adaptive Control Design* (John Wiley & Sons, Inc., 1995).
- K. S. Narendra, A. M. Annaswamy, *Stable Adaptive Systems* (Courier Corporation, 2012).
- P. A. Farrell, M. M. Polycarpou, *Adaptive Approximation Based Control* (John Wiley & Sons, Ltd., 2006); <https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471781819.fmatter>.
- K. A. Wise, E. Lavretsky, N. Hovakimyan, Adaptive control of flight: theory, applications, and open problems, in *Proceedings of the 2006 American Control Conference* (IEEE, 2006).
- X. Shi, P. Spieler, E. Tang, E.-S. Lupu, P. Tokumaru, S.-J. Chung, Adaptive Nonlinear Control of Fixed-Wing VTOL with Airflow Vector Sensing, in *2020 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2020), pp. 5321–5327.
- A. Rahimi, B. Recht, Random features for large-scale kernel machines, in *Proceedings of the 20th International Conference on Neural Information Processing Systems* (Advances in neural information processing systems, 2007), pp. 1177–1184.
- S. Lale, K. Azizzadenesheli, B. Hassibi, A. Anandkumar, Model Learning Predictive Control in Nonlinear Dynamical Systems, in *Proceedings of the 2021 60th IEEE Conference on Decision and Control (CDC)* (IEEE, 2021), pp. 757–762. ISSN: 2576-2370.
- J. Nakanishi, J. Farrell, S. Schaal, A locally weighted learning composite adaptive controller with structure adaptation, in *IEEE/RSI International Conference on Intelligent Robots and Systems* (IEEE, 2002), vol. 1, pp. 882–889.
- F.-C. Chen, H. K. Khalil, Adaptive control of a class of nonlinear discrete-time systems using neural networks. *IEEE Trans. Automatic Control* **40**, 791–801 (1995).
- E. N. Johnson, A. J. Calise, Limited authority adaptive flight control for reusable launch vehicles. *J. Guid. Control Dynam.* **26**, 906–913 (2003).
- K. S. Narendra, S. Mukhopadhyay, Adaptive control using neural networks and approximate models. *IEEE Trans. Neural Netw.* **8**, 475–485 (1997).
- M. Bisheban, T. Lee, Geometric adaptive control with neural networks for a quadrotor in wind fields. *IEEE Trans. Control Syst. Technol.* **29**, 1533–1548 (2021).
- Y. LeCun, Y. Bengio, G. Hinton, Deep learning. *Nature* **521**, 436–444 (2015).
- C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in *Proceedings of the 34th International Conference on Machine Learning (PMLR, 2017)*, pp. 1126–1135.
- T. M. Hospedales, A. Antoniou, P. Micaelli, A. J. Storkey, Meta-learning in neural networks: A survey, in *IEEE Transactions on Pattern Analysis and Machine Intelligence* (IEEE, 2021), pp. 1.
- A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, C. Finn, Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. arXiv: 1803.11347 [cs.LG] (2018).
- X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, J. Tan, Rapidly adaptable legged robots via evolutionary meta-learning, in *Proceedings of the 2020 IEEE/RSI International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2020), pp. 3769–3776.
- S. Belkhal, R. Li, G. Kahn, R. McAllister, R. Calandra, S. Levine, Model-based meta-reinforcement learning for flight with suspended payloads. *IEEE Robot. Autom. Lett.* **6**, 1471–1478 (2021).
- C. D. McKinnon, A. P. Schoellig, Meta learning with paired forward and inverse models for efficient receding horizon control. *IEEE Robot. Autom. Lett.* **6**, 3240–3247 (2021).
- I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, P. Abbeel, Model-based reinforcement learning via meta-policy optimization, in *Conference on Robot Learning* (PMLR, 2018), pp. 617–629.
- M. O'Connell, G. Shi, X. Shi, S.-J. Chung, Meta-learning-based robust adaptive flight control under uncertain wind conditions. arXiv: 2103.01932 [cs.RO] (2021).
- S. M. Richards, N. Azizan, J.-J. E. Slotine, M. Pavone, Adaptive-control-oriented meta-learning for nonlinear systems. arXiv:2103.04490 [cs.RO] (2021).

39. M. Peng, B. Zhu, J. Jiao, Linear representation meta-reinforcement learning for instant adaptation. arXiv: 2101.04750 [cs.LG] (2021).
40. J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, M. Hutter, Learning quadrupedal locomotion over challenging terrain. *Sci. Robot.* **5**, eabc5986 (2020).
41. J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, in *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2017), pp. 23–30.
42. F. Ramos, R. C. Possas, D. Fox, Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators. arXiv: 1906.01728 [cs.RO] (2019).
43. D. Morgan, G. P. Subramanian, S.-J. Chung, F. Y. Hadaegh, Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming. *Int. J. Robot. Res.* **35**, 1261–1285 (2016).
44. X. Shi, K. Kim, S. Rahili, S.-J. Chung, Nonlinear control of autonomous flying cars with wings and distributed electric propulsion, in *Proceedings of the 2018 IEEE Conference on Decision and Control (CDC)* (IEEE, 2018), pp. 5326–5333.
45. Y. K. Nakka, A. Liu, G. Shi, A. Anandkumar, Y. Yue, S.-J. Chung, Chance-constrained trajectory optimization for safe exploration and learning of nonlinear systems. *IEEE Robot. Autom. Lett.* **6**, 389 (2021).
46. A. Loquercio, E. Kaufmann, R. Ranftl, M. Muller, V. Koltun, D. Scaramuzza, Learning high-speed flight in the wild. *Sci. Robot.* **6**, eabg5810 (2021).
47. K. Kim, P. Spieler, E.-S. Lupu, A. Ramezani, S.-J. Chung, A bipedal walking robot that can fly, slackline, and skateboard. *Sci. Robot.* **6**, eabf8136 (2021).
48. Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, V. Lempitsky, Domain-adversarial training of neural networks. *J. Mach. Learn. Res.* **17**, 2096 (2017).
49. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets. *Adv. Neural Inform. Process. Syst.* **27**, (2014).
50. R. E. Kalman, R. S. Bucy, New results in linear filtering and prediction theory. *J. Basic Eng.* **83**, 95 (1961).
51. R. M. Murray, Z. Li, S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation* (CRC Press, ed. 1, 2017).
52. L. Trefethen, Multivariate polynomial approximation in the hypercube. *Proc. Am. Math. Soc.* **145**, 4837–4844 (2017).
53. D. Yarotsky, Error bounds for approximations with deep relu networks. *Neural Netw.* **94**, 103–114 (2017).
54. L. Dieci, T. Eirola, Positive definiteness in the numerical solution of Riccati differential equations. *Numerische Mathematik* **67**, 303–313 (1994).
55. R. E. Kalman, A new approach to linear filtering and prediction problems. *J. Basic Eng.* **82**, 35–45 (1960).
56. H. K. Khalil, *Nonlinear Systems, 3rd Edition* (Prentice Hall, 2002).
57. Multicopter PID Tuning Guide (Advanced/Detailed) I PX4 User Guide.

Acknowledgments: 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:** S.-J.C. and Y.Y. directed the research activities. G.S. and M.O. designed and implemented the metalearning algorithm under the guidance of Y.Y., K.A., A.A., and S.-J.C., while the last-layer adaptation idea was started with a discussion by G.S., M.O., X.S., and S.-J.C. M.O. and G.S. designed and implemented the adaptive control algorithm with inputs from S.-J.C. and X.S. M.O. and G.S. performed experiments and evaluated the results. M.O. conducted the theoretical analysis of the meta-learning based adaptive controller with input from S.-J.C., G.S., and X.S. G.S. analyzed the learning algorithm with feedback from Y.Y., K.A., A.A., and S.-J.C. G.S. and M.O. created all the figures and videos with input from the other authors. 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. We have provided the machine learning model training code, training data, and experimental data at github.com/aerobotics/neural-fly.

Submitted 11 October 2021
 Accepted 12 April 2022
 Published 4 May 2022
 10.1126/scirobotics.abm6597

Neural-Fly enables rapid learning for agile flight in strong winds

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

Sci. Robot. 7 (66), eabm6597. DOI: 10.1126/scirobotics.abm6597

View the article online

<https://www.science.org/doi/10.1126/scirobotics.abm6597>

Permissions

<https://www.science.org/help/reprints-and-permissions>

Use of this article is subject to the [Terms of service](#)

Science Robotics (ISSN 2470-9476) is published by the American Association for the Advancement of Science, 1200 New York Avenue NW, Washington, DC 20005. The title *Science Robotics* is a registered trademark of AAAS.

Copyright © 2022 The Authors, some rights reserved; exclusive licensee American Association for the Advancement of Science. No claim to original U.S. Government Works