SUPPLEMENTARY TEXT

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.

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 scale 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 \( \tilde{f}(x, w) : [-1, 1]^n \times [-1, 1]^m \to \mathbb{R} \) can be split into a \( w \)-invariant part \( \tilde{\phi}(x) \) and a \( w \)-dependant part \( \tilde{a}(w) \) in the structure \( \tilde{\phi}(x) \tilde{a}(w) \) with arbitrary precision \( \epsilon \), where \( \tilde{\phi}(x) \) and \( \tilde{a}(w) \) are two polynomials. Further, the dimension of \( \tilde{a}(w) \) only scales polylogarithmically with \( 1/\epsilon \).

We first introduce the following multivariate polynomial approximation lemma in the hypercube proved in (52).

Lemma 2. (Multivariate polynomial approximation in the hypercube) Let \( \tilde{f}(x, w) : [-1, 1]^n \times [-1, 1]^m \to \mathbb{R} \) be
a smooth function of \([x, w] \in [-1, 1]^{n+m}\) for \(n, m \geq 1\). Assume \(\tilde{f}(x, w)\) 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 \(\tilde{f}\) has a uniformly and absolutely convergent multivariate Chebyshev series

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

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

\[b_s = O((1+t)^{-\|s\|^2}).\]

Note that this lemma shows that the truncated Chebyshev expansions

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

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

**Theorem 3.** \(\tilde{f}(x, w)\) is a function satisfying the assumptions in Lemma 2. For any \(\epsilon > 0\), there exist \(h \in \mathbb{Z}^+\), and two Chebyshev polynomials \(\tilde{\phi}(x) : [-1, 1]^n \to \mathbb{R}^{1 \times h}\) and \(\tilde{a}(w) : [-1, 1]^m \to \mathbb{R}^{h \times 1}\) such that

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

and \(h = O((\log(1/\epsilon))^{n+m})\).

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

\[g(x, k, l) = g(x_1, \cdots, x_n, k_1, \cdots, k_n, l_1, \cdots, l_m) = b_{k_1, \ldots, k_n, l_1, \ldots, l_m} T_{k_1}(x_1) \cdots T_{k_n}(x_n)\]

\[g(w, l) = g(w_1, \cdots, w_m, l_1, \cdots, l_m) = T_{l_1}(w_1) \cdots T_{l_m}(w_m)\]

Then we have

\[C_p(x, w) = \sum_{k_1, \cdots, k_n=0}^{p} \sum_{l_1, \cdots, l_m=0}^{p} g(x, k_1, \cdots, k_n, l_1, \cdots, l_m) g(w, l_1, \cdots, l_m)\]
Then we rewrite $C_p$ as $C_p(x, w) = \bar{\phi}(x)\bar{a}(w)$:

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

$\bar{a}(w)$.

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

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

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 (53), 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 $\bar{a}$ 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 $f(x, w)$.

**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 $\sum_{i \in B} \| y_k^{(i)} - f(x_k^{(i)})a^* \|^2$ (see Line 7 in Algorithm 1), which reflects how well $\phi$ 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 $\phi$ 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 \( \phi \) 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 (49). (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 \( \alpha \) 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 \( \alpha^* \) between wind conditions. However, if \( \alpha \) is too high it may degrade the prediction performance, so we recommend using relatively small value for \( \alpha \) 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 \( (\alpha^{(1)} \) and \( \alpha^{(2)} \) in the first line in fig. S2 shown below). However, because of the expressiveness of DNNs, \( \phi \) 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, \( \phi_1 \) and \( \phi_2 \) are triggered if \( x \) is in \( P_1 \) and \( P_2 \), respectively \( (1_{x \in P_1} \) and \( 1_{x \in P_2} \) are indicator functions), such that the latent variable \( \alpha \) is identical in both wind conditions. Such an overfitted \( \phi \) is not robust and not generalizable: for example, if the drone flies to \( P_1 \) in wind condition 2, the wrong mode \( \phi_1 \) will be triggered.

The key idea to tackle this challenge is to encourage diversity in the latent space, which is why we introduced a discriminator in DAIML. Figure 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 covariance matrix $P$. During the development of our algorithm, we observed that a naive one-step Euler integration of the continuous time adaptation law would sometimes result $P$ becoming non-positive-definite due to a large $\dot{P}$ magnitude and a coarse integration step size (see (54) 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 $\hat{a}$ and $P$ are regularized. Then, in the update step in Eq. (18) and (19), $P$ and $\hat{a}$ are updated according to the gain in Eq. (17). This mirrors a discrete Kalman filter implementation (55) 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 $\phi$ are constant between time steps ensuring the positive definiteness of $P$.

$$\begin{align*}
\hat{a}_{t_k^-} &= (1 - \lambda \Delta t_k) \hat{a}_{t_k^+} \\
P_{t_k^-} &= (1 - \lambda \Delta t_k)^2 P_{t_k^+} + Q \Delta t_k \\
K_{t_k} &= P_{t_k} \phi_{t_k}^T \left( \phi_{t_k} P_{t_k} \phi_{t_k}^T + R \Delta t_k \right)^{-1} \\
\hat{a}_{t_k^+} &= \hat{a}_{t_k^-} - K_{t_k} \left( \phi_{t_k} \hat{a}_{t_k^-} - y_{t_k} \right) - P_{t_k^-} \phi_{t_k}^T s_{t_k} \\
P_{t_k^+} &= (I - K_{t_k} \phi_{t_k}) P_{t_k^-} (I - K_{t_k} \phi_{t_k})^T + K_{t_k} R \Delta t_k K_{t_k}^T
\end{align*}$$

**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, $\|s; \hat{a}\|$, 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, $\hat{q}$ 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 \( \hat{a} = \hat{a} - a \), we find

\[
M \dot{s} + (C + K)s = -\phi \hat{a} + d
\]

We can use the Lyapunov function \( V = s^T Ms \) under the assumption of bounded \( \hat{a} \) to show that

\[
\lim_{t \to \infty} \|s\| \leq \frac{\sup_t \|d - \phi \hat{a}\| \lambda_{\max}(M)}{\lambda_{\min}(K) \lambda_{\min}(M)}
\]

Taking this results alone, one might expect that any online estimator or learning algorithm will lead to good performance. However, the boundedness of \( \hat{a} \) is not guaranteed; Slotine and Li discuss this topic thoroughly (15). 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 \( \hat{a} \), and the adaptation law Eq. (8) and (9), the combined velocity and parameter-error closed-loop dynamics are given by

\[
\begin{bmatrix}
M & 0 \\
0 & P^{-1}
\end{bmatrix}
\begin{bmatrix}
s \\
\hat{a}
\end{bmatrix} +
\begin{bmatrix}
C + K & \phi \\
-\phi^T & \phi^T R^{-1} \phi + \lambda P^{-1}
\end{bmatrix}
\begin{bmatrix}
s \\
\hat{a}
\end{bmatrix} =
\begin{bmatrix}
d \\
\phi^T R^{-1} \bar{\epsilon} - P^{-1} \lambda \hat{a} - P^{-1} \dot{\hat{a}}
\end{bmatrix}
\]

\[
\frac{d}{dt}(P^{-1}) = -P^{-1} PP^{-1} = P^{-1} (2\lambda P - Q + P \phi^T R^{-1} \phi P) P^{-1}
\]

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 (54) show the slightly weaker result that that \( P \) is positive definite and finite when \( \phi \) is bounded under the looser assumption \( Q \succeq 0 \). Following the proof from (54) 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 $\begin{bmatrix} s \\ \hat{a} \end{bmatrix}$ exponentially converges to the bound given in Eq. (24) with rate $\alpha$.

$$\lim_{t \to \infty} \| \begin{bmatrix} s \\ \hat{a} \end{bmatrix} \| \leq \frac{1}{\alpha \lambda_{\min}(\mathcal{M})} \left( \sup_{t} \| d \| + \sup_{t} (\| \phi^T R^{-1} \hat{\epsilon} \|) + \lambda_{\max}(P^{-1}) \sup_{t} (\| \lambda a + \hat{a} \|) \right)$$

(24)

where $\alpha$ and $\mathcal{M}$ are functions of $\phi, R, Q, K, M$ and $\lambda$, 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 $V$ given by

$$V = \begin{bmatrix} s \\ \hat{a} \end{bmatrix}^T \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \hat{a} \end{bmatrix}$$

(25)

This Lyapunov function has the derivative

$$\dot{V} = 2 \begin{bmatrix} s \\ \hat{a} \end{bmatrix}^T \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} \dot{s} \\ \dot{\hat{a}} \end{bmatrix} + 2 \begin{bmatrix} s \\ \hat{a} \end{bmatrix}^T \begin{bmatrix} M & 0 \\ 0 & \frac{d}{dt} (P^{-1}) \end{bmatrix} \begin{bmatrix} s \\ \hat{a} \end{bmatrix}$$

(26)

$$= -2 \begin{bmatrix} s \\ \hat{a} \end{bmatrix}^T \begin{bmatrix} C + K & \phi^T R^{-1} \phi + \lambda P^{-1} \\ -\phi^T & \phi^T R^{-1} \phi + \lambda P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \hat{a} \end{bmatrix} + 2 \begin{bmatrix} s \\ \hat{a} \end{bmatrix}^T \begin{bmatrix} \phi^T R^{-1} \hat{\epsilon} - P^{-1} \lambda a - P^{-1} \hat{a} \\ \phi^T R^{-1} \hat{\epsilon} - P^{-1} \lambda a - P^{-1} \hat{a} \end{bmatrix}$$

(27)

$$+ \begin{bmatrix} s \\ \hat{a} \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 0 & 2 \lambda P^{-1} - P^{-1} Q P^{-1} + \phi^T R^{-1} \phi \end{bmatrix} \begin{bmatrix} s \\ \hat{a} \end{bmatrix}$$

(28)

$$= -2 \begin{bmatrix} s \\ \hat{a} \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 0 & 2 \phi^T R^{-1} \phi + P^{-1} Q P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \hat{a} \end{bmatrix} + 2 \begin{bmatrix} s \\ \hat{a} \end{bmatrix}^T \begin{bmatrix} \phi^T R^{-1} \hat{\epsilon} - P^{-1} \lambda a - P^{-1} \hat{a} \\ \phi^T R^{-1} \hat{\epsilon} - P^{-1} \lambda a - P^{-1} \hat{a} \end{bmatrix}$$

(29)

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

$$- \begin{bmatrix} 2K & 0 \\ 0 & \phi^T R^{-1} \phi + P^{-1} Q P^{-1} \end{bmatrix} \leq -2\alpha \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix}$$

(32)

for all $t$.

Define an upper bound for the disturbance term $D$ as

$$D = \sup_{t} \left\| \begin{bmatrix} \phi^T R^{-1} \hat{\epsilon} - P^{-1} \lambda a - P^{-1} \hat{a} \end{bmatrix} \right\|$$

(33)
and define the function $\mathcal{M}$,

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

(34)

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

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

(35)

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

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

(36)

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

(37)

By the Comparison Lemma (56),

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

(38)

and the stacked state exponentially converges to the ball

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

(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 $\alpha$, and $s = \dot{\tilde{q}} + \Lambda \tilde{q}$, then

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

(40)

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

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

(41)
Proof. From the Comparison Lemma (56), we can easily show Eq. 40. This can be further reduced as follows.

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

(42)

\[
\leq e^{-\lambda_{\min}(\Lambda) t} \| \tilde{q}(0) \| + A \frac{e^{-\alpha t} - e^{-\lambda_{\min}(\Lambda) t}}{\lambda_{\min}(\Lambda) - \alpha} + B \frac{1 - e^{-\lambda_{\min}(\Lambda) t}}{\alpha \lambda_{\min}(\Lambda)}
\]  

(43)

Taking the limit, we arrive at Eq. 41. 

\[\square\]

With the following corollary, we will justify that \( \alpha \) 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 \( \mathbf{a} \), that is, we expect a non-zero \( \phi \).

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

\[
\lim_{t \to \infty} \begin{bmatrix} s^T \\ \hat{a}^T \end{bmatrix} = \sup_{\| \mathbf{d} \|} \left( \lambda_{\max}(\mathbf{P}^{-1}) \sup_{\| \mathbf{a} \|} \| \mathbf{a} + \hat{a} \| \right) \min_{\lambda, \lambda_{\min}(\mathbf{K})/\lambda_{\max}(\mathbf{M})} \lambda_{\min}(\mathbf{M})
\]  

(44)

Proof. Assuming \( \phi \equiv 0 \) immediately leads to \( \alpha \) of

\[
\alpha = \min \left( \frac{1}{2} \lambda_{\min}(\mathbf{P}^{-1} \mathbf{Q}), \frac{\lambda_{\min}(\mathbf{K})}{\lambda_{\max}(\mathbf{M})} \right)
\]  

(45)

\( \phi \equiv 0 \) also simplifies the \( \dot{\mathbf{P}} \) equation to a stable first-order differential matrix equation. By integrating this simplified \( \dot{\mathbf{P}} \) equation, we can show \( \mathbf{P} \) exponentially converges to the value \( \mathbf{P} = \frac{\mathbf{Q}}{2\alpha} \). This leads to bound in Eq. 44. 

\[\square\]

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 \( \mathbf{f} \) 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 \( \mathbf{f} \) 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

\[
\lim_{t \to \infty} \left\| \begin{bmatrix} s \\ \dot{a} \end{bmatrix} \right\| \leq \frac{(1 + r^{-1}) \sup_{t} \| \mathbf{f} - \mathbf{a} \| + \epsilon/r}{\lambda_{\min}(K) \lambda_{\min}(M)} \lambda_{\max}(M)
\]  

(46)

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

\[
\frac{dp}{dt} = q - p^2/r
\]

(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

\[
\lim_{t \to \infty} \left\| \begin{bmatrix} s \\ \dot{a} \end{bmatrix} \right\| \leq \frac{D \lambda_{\max}(M)}{\lambda_{\min}(K) \lambda_{\min}(M)}
\]

(48)

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

\[
D \leq (1 + r^{-1}) \sup_{t} \| \mathbf{f} - \mathbf{a} \| + \epsilon/r
\]

(49)

Section S6  Gain Tuning

The attitude controller was tuned following the method in (57). The gains for all of the position controllers tested were tuned on a step input of 1 m in the x-direction. The proportional (P) and derivative (D) gains were tuned using the baseline nonlinear controller for good rise time with minimal overshoot or oscillations. The same P and D gains were used across all methods.

The integral and adaptation gains were tuned separately for each method. In each case, the gains were increased to minimize response time until we observed having large overshoot, noticeably jittery, or oscillatory behavior. For \( \mathcal{L}_1 \) and INDI, this gave a first-order filters with a cutoff frequency of 5 Hz. For each of the Neural-Fly methods, we used \( R = rI \) and \( Q = qI \), where \( r \) and \( q \) are 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 \( \lambda \) values of 0.01.
**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 trajectory 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 $L_1$, 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 cm. This is based on a comparison of our two different localization data sources. The first is the OptiTrack motion capture system, which uses several infrared motion tracking cameras and reflective markers on the drone to produce a delayed measurement the position and orientation of the vehicle. The PX4 flight controller runs an onboard extended Kalman filter (EKF) to fuse the OptiTrack measurements with onboard inertial measurement unit (IMU) measurements to produce position, orientation, velocity, and angular rate estimates. In offline analysis, we correct for the delay of the OptiTrack system, and compare the position outputs of the OptiTrack system and the EKF. Typical results are shown in fig. 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 cm. Thus, we estimate that the localization precision has a standard deviation of about 1.0 cm.
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.

Figure S2: **Importance of domain-invariant representation.**
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 $K_i \int \hat{p} dt$, 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.
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.

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

Table S1: **Drone configuration details** Configurations of the custom built drone and the Intel Aero drone with propeller guards.
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.

<table>
<thead>
<tr>
<th></th>
<th>Neural-Fly</th>
<th>INDI (4)</th>
<th>Differentially flat linear drag (2)</th>
<th>Gaussian Process MPC (11)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flight computer</td>
<td>Raspberry Pi 4</td>
<td>–</td>
<td>laptop</td>
<td>laptop</td>
</tr>
<tr>
<td>Flight controller</td>
<td>Pixhawk 4</td>
<td>STM32H7 (400 MHz)</td>
<td>Raceflight Revolt</td>
<td>Raceflight Revolt</td>
</tr>
<tr>
<td>Flight controller firmware</td>
<td>PX4</td>
<td>custom</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Mass [kg]</td>
<td>2.53</td>
<td>0.609</td>
<td>0.610</td>
<td>0.8</td>
</tr>
<tr>
<td>Total width [cm]</td>
<td>85</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Propeller diameter [in]</td>
<td>11</td>
<td>5</td>
<td>6</td>
<td>?</td>
</tr>
<tr>
<td>Motor Spacing [cm]</td>
<td>39*</td>
<td>18</td>
<td>–</td>
<td>?</td>
</tr>
<tr>
<td>Thrust-to-weight ratio [-]</td>
<td>2.2</td>
<td>?</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Motion capture frequency [Hz]</td>
<td>100</td>
<td>360</td>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>MPC control frequency [Hz]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>50</td>
</tr>
<tr>
<td>Position control frequency [Hz]</td>
<td>50</td>
<td>?</td>
<td>55</td>
<td>?</td>
</tr>
<tr>
<td>Attitude control frequency [Hz]</td>
<td>&lt;1000</td>
<td>2000</td>
<td>4000</td>
<td>?</td>
</tr>
<tr>
<td>Motor speed feedback</td>
<td>No</td>
<td>Optical encoders (5000 Hz)</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

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

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

| Architecture of $\phi$ net | $11 \rightarrow 50 \rightarrow 60 \rightarrow 50 \rightarrow 4$ with ReLU activation functions |
| Architecture of $h$ net     | $4 \rightarrow 128 \rightarrow 6$ 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 $\phi$ | 0.0005 |
| Learning rate for training $h$ | 0.001 |
| Discriminator training frequency $\eta$ | 0.5 |
| Normalization constant $\gamma$ | 10 |
| The degree of regularization $\alpha$ | 0.1 |

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