

A Learning-based Quadcopter Controller with Extreme Adaptation

Dingqi Zhang¹, Antonio Loquercio², Jerry Tang¹, Ting-Hao Wang¹,
Jitendra Malik³, and Mark W. Mueller¹

Abstract—This paper introduces a learning-based low-level controller for quadcopters, which adaptively controls quadcopters with significant variations in mass, size, and actuator capabilities. Our approach leverages a combination of imitation learning and reinforcement learning, creating a fast-adapting and general control framework for quadcopters that eliminates the need for precise model estimation or manual tuning. The controller estimates a latent representation of the vehicle’s system parameters from sensor-action history, enabling it to adapt swiftly to diverse dynamics. Extensive evaluations in simulation demonstrate the controller’s ability to generalize to unseen quadcopter parameters, with an adaptation range up to 16 times broader than the training set. In real-world tests, the controller is successfully deployed on quadcopters with mass differences of 3.7 times and propeller constants varying by more than 100 times, while also showing rapid adaptation to disturbances such as off-center payloads and motor failures. These results highlight the potential of our controller to simplify the design process and enhance the reliability of autonomous drone operations in unpredictable environments. Video and code are at: https://github.com/muellerlab/xadapt_ctrl

I. INTRODUCTION

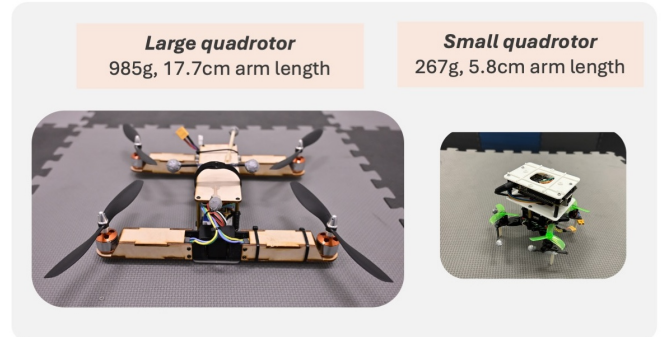
The agile nature of quadcopters and the necessity for precise control in dynamic environments create a unique context for exploring control strategies. Model-based controllers for quadcopters generally rely on estimates of the vehicle’s properties, including inertia, motor constants, and other parameters. Notable examples include sliding mode control [1] and PID controllers [2]. Once these parameters are estimated, the controller typically requires iterative tuning through successive experiments to refine its performance. However, inaccuracies in parameter estimation can directly lead to execution errors in controller commands. Furthermore, any modification to the vehicle, such as attaching an extra payload, could lead to suboptimal performance without repeating the estimation and tuning processes. Such significant engineering effort could be eased with a universal controller that does not require specialized tuning.

In this work, we propose a learning-based low-level controller designed to control a variety of quadcopters with notable differences in mass, size, propellers, and motors. Our controller is also capable of rapidly adapting to unknown in-flight disturbances such as off-center payloads, loss of efficiency in motors, and wind.

A. Related Work

The development of fixed-parameter controllers, while foundational, is inherently limited by their lack of real-time

The authors are with the ¹High Performance Robotics Lab, Dept. of Mechanical Engineering, UC Berkeley, ² the University of Pennsylvania (most work done while at UC Berkeley), and ³ Dept. of Electrical Engineering and Computer Science, University of California at Berkeley. Contact at {dingqi, loquercio, jerrytang, wtyng, malik, mwm}@berkeley.edu



(a) Hardware Details



(b) Deployment

Fig. 1: Our adaptive controller can control quadcopters with large difference in parameters such as mass, arm length, and actuators while also show disturbance rejection. (a) Two vehicles flown with our controller, with mass differing by a factor of 3.68, arm length by 3.1 and motor constant by more than 100x. (b) Demonstration of our controller on these two vehicles for the task of tracking trajectories under disturbances including off-center payload and wind. We use a single control policy across different drones and tasks, which is deployed without any vehicle-specific modifications.

adaptivity to model uncertainties and disturbances. Adaptive control techniques were introduced to address these unpredicted variations in a system. One of the initial contributions in this field was the model reference adaptive controller (MRAC), an extension of the well-known MIT-rule [3]. The empirical success of MRAC led to the development of \mathcal{L}_1 control [4], [5], which offers a promising solution by estimating the differences between the nominal state transitions predicted by the reference model and those observed in practice. Such differences are

compensated by allocating a control authority proportional to the disturbance, effectively driving the system back to its reference behavior.

However, the performance of the classic \mathcal{L}_1 formulation degrades when the observed transitions deviate greatly from the (usually linear) reference model. This limitation is critical in scenarios involving large variations between different quadcopters’ dynamics, as the underlying assumptions on locally linear disturbances are generally not fulfilled. Some studies have discussed practical considerations regarding its implementation. For instance, high adaptive gains in certain scenarios may introduce numerical challenges and sensitivity to unmodeled dynamics [6]. It has also been noted that, under specific conditions, the closed-loop behavior of \mathcal{L}_1 controllers can resemble that of linear proportional-integral (PI) controllers [7], [8], which may limit the expression of its adaptive features. To address these issues, recent research has explored combining \mathcal{L}_1 adaptive control with nonlinear online optimization techniques, such as model predictive control (MPC) [9], [10], [11]. These methods have achieved impressive results, but still require explicit and accurate knowledge of the reference model for adaptation.

Beyond \mathcal{L}_1 , advanced nonlinear adaptive controllers have been developed to enable agile quadcopter maneuvers in the presence of uncertainty. Notable approaches include geometric adaptive tracking control on SE(3) [12] and adaptive incremental nonlinear dynamic inversion (INDI) [13]. These controllers improve performance by directly incorporating nonlinear dynamics and real-time uncertainty estimation into the control design. A nominal model prior is first estimated and then used for subsequent compensation of dynamic variations, enhancing robustness during high-precision maneuvers.

Recent advances in data-driven control strategies have shown promising results for quadcopter stabilization [14], [15], or waypoint tracking flight [16], [17], as well as agile racing against human pilots [18]. Model-free reinforcement learning in [18] has demonstrated impressive adaptability to unmodeled disturbances, such as blade flapping effects. Combining data-driven methods with model-based control designs has also been proposed to leverage the guaranteed adaptivity and robustness offered by model-based control. For instance, some studies have learned policies from model-based methods like Robust-Tube MPC through imitation learning [19], [20], [21]. Another approach is augmenting the learned controller with classical adaptive control designs during deployment for fast disturbance estimation and online adaptation [22], [23]. Despite these advancements, these methods remain tailored to specific platforms. Transferring the same controller to another vehicle typically requires retraining or fine-tuning the policy, along with data collection for the new vehicle.

Zero-shot adaptation across different vehicles has been demonstrated on quadrupeds [24], highlighting the versatility of learning-based methods. However, this generalized control relies on existing internal motor control loops rather than directly adapting at the motor level. In the case of quadcopters, the variation in actuators between different vehicles is particularly significant, with motor constants potentially differing by orders of magnitude. Consequently, effective adaptation

across different quadcopter platforms must address motor-level differences directly. Additionally, the high-frequency nature of motor control increases the risk of crashes due to inadequate adaptation. These factors, the substantial variation in actuators and the high-frequency nature of motor control, pose significant challenges to applying previous methods of adaptive trajectory control to the problem of extreme adaptation across quadcopters.

B. Our Contributions

In this work, we present a general framework for learning low-level adaptive controllers that are effective across a wide range of quadcopters. Similar to prior research in learning-based control for aerial robots [25], [17], [26], [27], [14], [28], we train the policy entirely in simulation using reinforcement learning and deploy it directly to the real world without fine-tuning (i.e., zero-shot deployment).

While previous works typically rely on slight parameter randomizations (up to 30%) around a nominal model [17], our approach must handle parameter variations thousands of times larger. This presents a significant challenge for reinforcement learning, as such a broad range of variations can hinder optimization convergence.

To address this challenge, we build on our earlier conference work on learning-based low-level control [29] and introduce three key technical innovations: (1) a dual training strategy that combines imitation learning from specialized model-based controllers and model-free reinforcement learning. This combination effectively handles the challenges of training a low-level controller due to its high-frequency nature and the low informational density of observations. (2) A specifically designed reward to provide the low-level controller with direct feedback for quick adjustments, allowing it to perform more agile maneuvers, and (3) a designed-informed domain randomization method to ensure that the variations in quadcopter designs during training are consistent with real-world constraints. These innovations eliminate constraints on slow flight and accurate state estimation, and widens the range of vehicles our policy can fly. Our approach significantly outperforms existing baselines. In addition, it enables the controller to adapt to out-of-distribution quadcopters in simulation up to 16x wider than the training set and to disturbances for which it was not explicitly trained, such as wind.

Our work shows a generalized controller for agile and robust flight of quadcopters with parameter differences of several orders of magnitude. Such large scale adaptability will help democratize the process of drone design by enabling users that lack modeling expertise to control custom-made vehicles.

II. METHOD

We present our methodology for learning an adaptive low-level controller for various quadcopters. A list of symbols and notations are given in Table I.

A. Control Structure

Cascade control systems are instrumental in managing complex dynamic systems by decomposing them into a hierarchy

TABLE I: List of Symbols

Notation (for arbitrary symbol X)			
X	Scalar quantity	\mathbf{x}	Vector quantity (e.g., ω)
X_{des}	Commanded quantity	X_{ref}	Quantity in reference trajectory
X_{min}	Minimum value	X_{max}	Maximum value
\dot{X}	Derivative of X w.r.t. time	\hat{X}	Estimated quantity
State Variables			
\mathbf{p}	Position	\mathbf{v}	Velocity
\mathbf{q}	Attitude (quaternion)		Yaw angle
ω	Angular velocity	$\boldsymbol{\tau}$	Torque
c_{Σ}	Mass-normalized total thrust	\mathbf{F}	Individual motor forces
\mathbf{a}	Individual motor speeds	\mathbf{a}_{pwm}	Individual motor Pulse width modulation (PWM) commands
Quadrotor Parameters			
l	Arm length	m	Mass
J	Mass moment of inertia (MMOI) matrix	C_d	Body drag coefficient
C_F	Propeller constant: thrust-to-motorspeed-squared ratio	C	Propeller constant: torque-to-thrust ratio
c	Size factor for randomization		The sampled quadrotor
M	Mixer matrix		
Learning Variables			
	Base policy (our low-level controller)		Adaptation module
	Intrinsics encoder	\mathbf{e}_t	Environmental parameters
\mathbf{z}_t	Intrinsics vector	\mathbf{x}_t	State vector
r_t	Reward		

of simpler nested subsystems. For example, in a two-layer system, the high-level component focuses on high-level tasks such as trajectory planning, while the low-level component acts as the inner control loop to execute the commands from the high level.

Our controller is designed to function as a low-level component within this modular control hierarchy. It translates high-level total thrust and angular velocity commands into individual motor speeds with adaptation to different quadcopters. The implementation of low-level adaptation abstracts away the physical complexities of the system from the high-level planner, allowing the latter to focus on high-level mission tasks. This flexibility also enables our controller to enhance non-adaptive high-level controllers, adding adaptability to disturbances and model mismatches and thereby improving overall system performance.

An overview of our training strategy is illustrated in Figure 2. In simulation, the controller learns to track diverse trajectories across various quadcopter models. The training follows a dual approach, combining reinforcement learning (RL) with imitation learning (IL) from an expert model-based controller. To ensure efficient and realistic training, we employ a design-informed randomization law to sample quadcopter parameters. This approach maintains adherence to real-world design constraints while enhancing training efficiency. The generated parameters are then used to update the expert model-based controller, ensuring it adapts its behavior as the simulated model changes.

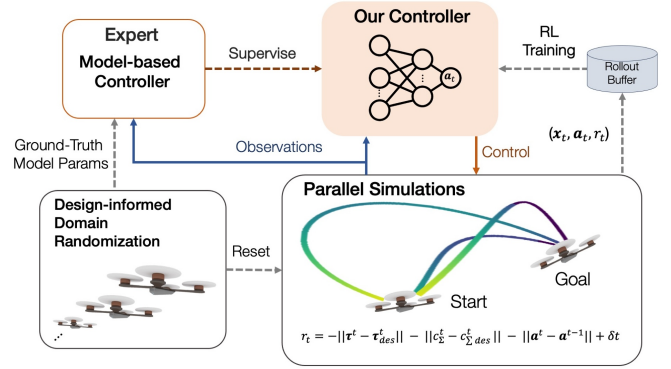


Fig. 2: An overview of the training process for our adaptive controller. The policy aims to track reference trajectories in simulation for various different quadcopters. The training framework employs a hybrid approach, combining reinforcement learning (RL) with imitation learning derived from a model-based controller. We use a design-informed randomization strategy to generate various quadcopters that adhere to general quadcopter design principles

The following subsections provide details on each component of the training process.

B. Training Framework

We adopt a learning-based framework that decouples policy learning into a control policy and a real-time estimator, commonly used in quadruped locomotion [30], [31]. The detailed training process is shown in Figure 3. We train in two phases. In the first, we train a low-level controller given access to ground-truth system parameters via RL and IL. Since we don't have such parameters in the real world, we use a second phase to learn an adaptation module. Such a module predicts the system parameter from a sensor-action history. The module is trained in simulation using supervised learning. During deployment, we can use the base policy and the adaptation module to achieve zero-shot adaptation.

Phase 1 Training. Our controller consists of a base policy, an intrinsics encoder, and an adaptation module. At time t , the base policy takes the current state $\mathbf{x}_t \in \mathbb{R}^8$ and the ground-truth intrinsics vector $\mathbf{z}_t \in \mathbb{R}^8$ to output the target motor speeds $\mathbf{a}_t \in \mathbb{R}^4$ for all individual motors. The intrinsics vector \mathbf{z}_t is a low dimensional encoding of the environment parameters $\mathbf{e}_t \in \mathbb{R}^{34}$, which consist of model parameters or external disturbances that are key to adaptive control. We use the intrinsics encoder to compress \mathbf{e}_t to \mathbf{z}_t . This gives us:

$$\mathbf{z}_t = \text{encoder}(\mathbf{e}_t) \quad (1)$$

$$\mathbf{a}_t = \text{policy}(\mathbf{x}_t; \mathbf{z}_t) \quad (2)$$

The current state \mathbf{x}_t includes the mass-normalized thrust c_{Σ} (\mathbb{R}), angular velocity ω (\mathbb{R}^3), commanded total thrust $c_{\Sigma; \text{des}}$ (\mathbb{R}) and commanded angular velocity ω_{des} (\mathbb{R}^3). The environmental parameter \mathbf{e}_t includes mass m , arm length l , propeller constants (torque-to-thrust ratio C and thrust-to-motorspeed-squared ratio C_F), the diagonal entries of MMOI matrix J (\mathbb{R}^3), body drag coefficients C_d (\mathbb{R}^3), maximum motor rotation, motor

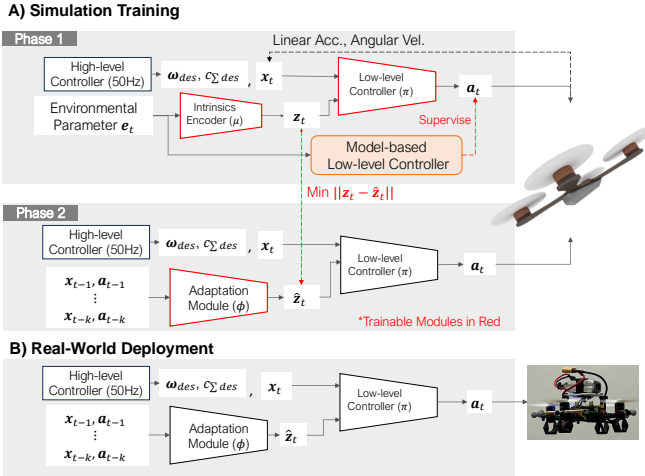


Fig. 3: The training (top) and the deployment architecture of our system (bottom). We train in two phases. In the first phase, we train a low-level controller via RL and IL. The policy takes the current state \mathbf{x}_t and the intrinsic vector \mathbf{z}_t , which is a compressed version of the environment parameters \mathbf{e}_t generated by the module. Since we cannot deploy this policy in the real world because the environment parameters \mathbf{e}_t are not available, we learn an adaptation module that takes the sensor-action history and directly predicts the intrinsic vector \mathbf{z}_t . This is done in phase two in simulation using supervised learning. We can finally deploy the base policy which takes as input the current state \mathbf{x}_t and the intrinsic vector $\hat{\mathbf{z}}_t$ predicted by the adaptation module.

effective factors (\mathbb{R}^4), mixer matrix M ($\mathbb{R}^{4 \times 4}$), payload mass, and external torque (\mathbb{R}^3), which results in an 34 dimensional vector. The mixer matrix M is the inverse of the allocation matrix. While the allocation matrix maps rotor forces to total thrust and torques, the mixer matrix maps desired thrust and torques to individual rotor forces, making it essential for motor control to determine the commands for individual rotors. A more detailed derivation is provided, for example, in [32].

The choice of 8 dimensions for the intrinsic vector \mathbf{z}_t is determined empirically through a binary search on the dimension of the intrinsic \mathbf{e}_t , aiming to optimize the learning performance of the policy. The latent representation of high-dimensional system parameters allows the base policy to adapt to variations in drone parameters, payloads, and disturbances such as external force or torque.

Phase 2 Training. During deployment, we do not have access to the environmental parameters \mathbf{e}_t and hence we cannot directly measure the intrinsic \mathbf{z}_t in the real world. Instead, we estimate it via the adaptation module, which uses the commanded action and the measured sensor readings from the latest k steps to estimate it online during deployment as (3). We can train this adaptation module in simulation using supervised learning because we have access to the ground truth intrinsic \mathbf{z}_t . We minimize the mean squared error loss $\|\mathbf{z} - \hat{\mathbf{z}}\|^2$ when $\hat{\mathbf{z}}$ is estimated using sensor-action history of the vehicle tracking trajectories generated with randomized motion primitives [33]. The random trajectory tracking task can provide a set of rich excitation signals for the adaptation module to estimate $\hat{\mathbf{z}}$ using the sensor-action history.

The estimated $\hat{\mathbf{z}}_t$ along with the current state \mathbf{x}_t is fed into

our base policy to output motor speed during deployment as (4). More concretely,

$$\hat{\mathbf{z}}_t = \mathbf{x}_{t-k:t-1} \mathbf{a}_{t-k:t-1} \quad (3)$$

$$\mathbf{a}_t = (\mathbf{x}_t; \hat{\mathbf{z}}_t) \quad (4)$$

C. Reward Design

The adaptive base policy functions as a low-level controller within the control hierarchy, capable of tracking arbitrary high-level commands irrespective of the vehicle being controlled. Our reward design should align with this objective by incentivizing the agent to track the specified reference high-level commands and penalizing crashes and oscillating motions.

The reward at time t is calculated as the sum of the following quantities:

- 1) Output Smoothness Penalty: $-\|\mathbf{a}^t - \mathbf{a}^{t-1}\|$
- 2) Survival Reward: t
- 3) Mass-normalized Thrust Tracking Deviation Penalty: $-\|\mathbf{c}_{\Sigma}^t - \mathbf{c}_{\Sigma,des}^t\|$
- 4) Torque Tracking Deviation Penalty: $-\|\mathbf{l}^t - \mathbf{l}_{des}^t\|$

Where t is the simulation step in the training episode. The mass-normalized thrust command $\mathbf{c}_{\Sigma,des}$ is given by the high-level controller along with the commanded angular velocity \mathbf{l}_{des} . The commanded torque \mathbf{l}_{des} is given by the rate control on the commanded angular velocity. In particular,

$$\mathbf{l}_{des} = K(\mathbf{l}_{des} - \mathbf{l}) \quad (5)$$

$$\mathbf{c}_{des} = J\mathbf{l}_{des} + \mathbf{l} \times (J\mathbf{l}) \quad (6)$$

Where K is a diagonal gain matrix, which we choose with values $K = \text{diag}(20; 20; 4)\text{s}^{-1}$ to effectively control the torques in roll, pitch, and yaw axes individually. The higher gains for roll and pitch (20) prioritize their control over yaw (4), due to their greater importance for flight stability and maneuverability.

The output oscillation penalty encourages smooth inputs, discouraging high-frequency control signals. The survival reward encourages the quadcopter to learn to fly longer until the end of the training episode. Finally, both tracking deviation penalties encourage the quadcopter to track the given high-level commands by matching its mass-normalized thrust and torque with the reference commands.

D. Guiding search by imitating a model-based controller

We implement the base policy and the intrinsic encoder as multi-layer perceptrons and jointly train them end-to-end in simulation. The training is done by an integration of IL and model-free RL. In our approach, the expert controller is a low-level proportional-derivative (PD) controller with access to the sampled vehicle's ground truth model parameters. It receives mass-normalized thrust and angular velocity commands from the high-level controller and computes the desired torque using (6). The resulting thrust and torque are linearly mapped to individual motor forces (\mathbf{F}_{des}) using the mixer matrix (M),

which are then converted into motor speeds ω_{exp} , representing the expert action.

$$F_{des} = M \ddot{r}_{des} \quad (7)$$

$$a_{exp} = \frac{F_{des}}{C_F} \quad (8)$$

where the square root operation is applied elementwise.

The key distinction from other work combining reinforcement and imitation learning [34], [35], [36] is that during each training episode, the ground-truth model parameters of randomized quadcopters are used to adapt the expert controller. This ensures that our base policy learns from an expert controller that dynamically adjusts its behavior whenever the quadcopter model changes. The IL loss minimizes the mean squared error loss on actions:

$$L_{IL}(\theta) = k a_{exp} - a_k^2 \quad (9)$$

Reinforcement learning maximizes the following expected return of the policy :

$$R_{RL}(\theta) = E_{p(\cdot|j)} \sum_{t=0}^{\infty} \gamma^t r_t \quad (10)$$

where $\tau = f(x_0; a_0; r_0); (x_1; a_1; r_1); \dots; g$ is the trajectory of the agent when executing the policy, and $p(\cdot|j)$ represents the likelihood of the trajectory under j .

We adaptively change the relative weight between these two losses. The overall training framework seeks to maximize the overall reward of the policy :

$$R(\theta) = (1 - \alpha) R_{RL}(\theta) - \alpha L_{IL}(\theta) \quad (11)$$

$$\alpha = e^{-0.001 t_{epoch}} \quad (12)$$

where the weight of the IL losses decays exponentially while the weight for RL increases inversely with training steps so that RL becomes dominant later in the training process. This training scheme enables rapid learning of the desired behavior from the expert controller at the beginning of training and generalization by RL in the later parts of training.

E. Quadcopter Parametric Randomization

The training of our adaptive policy requires a wide spectrum of quadcopters, a challenge that we address through a carefully crafted randomization process rather than relying on uniform sampling. We propose a randomization method that embodies key physical principles and design constraints of quadcopters, ensuring that the generated variations are physically plausible. Quadcopters follow a general design pattern, which typically involves a symmetric structure with four rotors positioned at the corners of a square frame. The size of a quadcopter is positively correlated with its mass, moment of inertia, and other properties, such as motor power and body drag coefficient. Our method follows the pattern in randomizing the quadcopter size and their respective dynamic characteristics, instead of simply varying parameters independently. In particular, we introduce a few key factors in quadcopter randomization which govern the variation of some other quadcopter body parameters. choice ensures that larger quadcopters, which typically require

TABLE II: Ranges of quadcopter and environmental parameters, along with the end states of the sampled trajectories from the initial conditions. Parameters without units are dimensionless.

Parameters	Training Range	Testing Range
Quadcopter Parameters		
Mass (kg)	[0.226, 0.950]	[0.205, 1.841]
Arm length (m)	[0.046, 0.200]	[0.040, 0.220]
MMOI around x, y (kg m ²)	[1.93e-4, 5.40e-3]	[1.73e-5, 2.27e-2]
MMOI around z (kg m ²)	[2.42e-4, 8.51e-3]	[2.10e-4, 3.40e-2]
Propeller constant:		
Torque-to-Thrust Ratio (m)	[0.0069, 0.0161]	[0.0051, 0.0170]
Payload (% of Mass)	[18, 40]	[18, 40]
Payload location from Center of Mass (% of Arm length)	[-50, 50]	[-50, 50]
Propeller Constant:	[3.88e-8, 8.40e-6]	[3.24e-9, 1.02e-4]
Thrust-to-Motorspeed-squared Ratio		
Body drag coef cient	[0, 0.74]	[0, 1.15]
Max. motor speed (rad/s)	[800, 8044]	[400, 10021]
Motor effectiveness factor	[0.7, 1.3]	[0.7, 1.3]
Motor time constant (s)	0.01	0.01
Sampled Trajectory End State from Initial Condition		
Position (m)	[-2, 2]	[-2, 2]
Velocity (m/s)	[-2, 2]	[-2, 2]
Acceleration (m/s ²)	[-2, 2]	[-2, 2]
Total Time (s)	[1, 5]	5

Size Factor. We introduce a size factor α , which uniformly scales the size and motor strength of the quadcopter. We randomly sample α from the range of [0, 1]. The arm length is linearly scaled with α with minimum and maximum values from the training range of Table II.

$$l = \alpha(l_{max} - l_{min}) + l_{min} \quad (13)$$

Assuming a constant density and proportional scaling in all dimensions, the mass of the quadcopter is proportional to its volume, which in turn scales with the cube of its arm length. Similarly, the moment of inertia, which depends on both the mass distribution and the distance from the axis of rotation, scales approximately with the fifth power of the arm length under these assumptions. The body drag coefficient, primarily influenced by the cross-sectional area the quadcopter presents to the air flow, scales with the square of the arm length.

We preserve the correlation by defining the mass, moment of inertia and body drag coefficient as

$$m = C_m (m_{max} - m_{min}) + m_{min} \quad (14)$$

$$C_m = \frac{l_{max}^3 - l_{min}^3}{l_{max}^3 - l_{min}^3}$$

$$J = C_J (J_{max} - J_{min}) + J_{min} \quad (15)$$

$$C_J = \frac{l_{max}^5 - l_{min}^5}{l_{max}^5 - l_{min}^5}$$

$$C_d = C_{C_d} (C_{d,max} - C_{d,min}) + C_{d,min} \quad (16)$$

$$C_{C_d} = \frac{l_{max}^2 - l_{min}^2}{l_{max}^2 - l_{min}^2}$$

with all minimum and maximum values from Table II.

To reflect the relationship between quadcopter size and motor strength, we choose to exponentially scale the motor thrust-to-motorspeed-squared ratio with the size factor. This design choice ensures that larger quadcopters, which typically require

more powerful motors, are equipped with appropriately scaled motor capabilities in our simulations.

$$C_F = C_{F_{\min}} \frac{C_{F_{\max}}}{C_{F_{\min}}}^c \quad (17)$$

Finally, all other parameters, such as maximum motor speed and propeller constant, are linearly scaled with the size factor. This factor and the associated randomization method ensure the correlation between quadcopter parameters, reducing the likelihood of generating physically unrealistic quadcopters (e.g., a very small and lightweight quadcopter equipped with overly powerful motors).

Noise. To ensure variability and allow for the fact that real systems will not perfectly follow these scaling rules, we introduce a uniformly distributed noise in the range of [-20%, 20%] to all parameters after they have been scaled with the size factor.

Motor Effectiveness Factor. We randomize the motor effectiveness factor for each of the four rotors so that the motor will produce a force different than expected. For each rotor the simulated motor speed is calculated by multiplying the intended speed by this factor. This is to simulate the motor's ineffectiveness due to battery voltage drop, a damaged propeller, or simply hardware variations.

External Disturbance. At a randomly sampled time during each episode, the parameters, including mass, inertia, and the center of mass, are again randomized. This is used to mimic the sudden variations in the quadcopter parameters due to a sudden disturbance caused by an off-center payload.

All our training and testing ranges in simulation are listed in Table II.

III. IMPLEMENTATION DETAILS

This section details the specific implementation of our approach, including the simulator for the training and evaluation of the policy, the hardware specifications for real-world experiments, and the neural network architectures with their training details.

Simulation Environment. We use the Flightmare simulator [37] to train and test our control policies. We implement the same high-level controller in [38] to generate high-level function share commands at the level of body rates and mass-normalized collective thrust for our low-level controller to track. It is designed as a cascaded linear acceleration controller with desired acceleration mimicking a spring-mass-damper system with natural frequency 2rad/s and damping ratio 0.7. The desired acceleration is then converted to the desired total thrust and the desired thrust direction, and the body rates computed from this as proportional to the attitude error angle with a time constant of 0.2s. The high-level controller's inputs are the platform's state (position, rotation, angular, and linear velocities) and the reference position, velocity, and acceleration from the generated trajectory at the simulated time point. The policy outputs individual motor speed commands, and we model the motors' response using a first-order system with a time constant of 10ms. Each RL episode lasts for a maximum of 500s of simulated time, with early termination if the vehicle loses

more than 10m from its starting height, or the quadcopter's body rate exceeds 10rad/s. The control frequency is 500Hz, which is also the simulation step. We additionally implement an measurement latency of 5ms.

Hardware Details. For all of our real-world experiments, we use two quadcopters, which differ in mass by a factor of 3.68, and in arm length by a factor of 3.1. The first one, which we name large quadrotor has a mass of 985g, a size of 17.7cm in arm length, a thrust-to-weight ratio of 3.62, a diagonal inertia matrix of $[0.004, 0.008, 0.012] \text{kg}^2$ (as expressed in the z-up body-fixed frame), and a maximum motor speed of 1000rad/s. The second one, small quadrotor has a mass of 267g, a size of 5.8cm in arm length, a thrust-to-weight ratio of 3.23, a diagonal inertia matrix of $[259\text{e-}6, 228\text{e-}6, 285\text{e-}6] \text{kg}^2$, and a maximum motor speed of 6994rad/s. For each of our platforms, we use a Qualcomm Robotics RB5 platform as the onboard computer which runs the high-level control at 50Hz and our deployed policy at 500Hz, and a mRo PixRacer as the flight control unit. We use as high-level a PID controller which takes as input the goal position, velocity, and acceleration and outputs the mass normalized collective trust and the body rates. An onboard Inertia Measurement Unit (IMU) measures the angular velocity and the acceleration of the robot, which is low-pass filtered to reduce noise and remove outliers. The high-level commands of the collective thrust and the body rates, and the low-level measurement of the angular rates and the acceleration are fed into the deployed policy as inputs. The policy outputs motor speed commands, which are sent to the PixRacer via the UART serial port and subsequently tracked by off-the-shelf electronic speed controllers. The real-world experiment is performed indoors with a motion capture system running. The motion capture is only used for evaluating the system's performance in experiments and for feedback to the high-level controller, but it does not provide any feedback to our policy.

Network Architecture and Training Procedure. The base policy is a 3-layer MLP with 256-dim hidden layers. This takes the drone state and the vector of intrinsics as input to produce motor speeds. The environment factor encoder is a 2-layer MLP with 128-dim hidden layers. The policy and the value network share the same factor encoding layer. The adaptation module projects the latest 100 state-action pairs into a 128-dim representation, with the state-action history initialized with zeros. We selected a window size of 100 as it provides good performance while keeping the network lightweight. Then, a 3-layer 1-D CNN convolves the representation across time to capture its temporal correlation. The input channel number, output channel number, kernel size and stride of each CNN layer are $[32, 32, 8, 4]$, $[32, 32, 5, 1]$, $[32, 32, 5, 1]$. The flattened CNN output is linearly projected to estimate the intrinsics vector z_t . For RL, we train the base policy and the adaptation module using PPO [39] for 100M steps in the environment. For supervised learning, we use the policy trained by Torch. We use the reward described in Section 6. Policy training takes approximately 1.5 hours on an ordinary desktop machine with 1 NVIDIA GeForce RTX 4060 GPU. We then train the adaptation module with supervised learning by rolling out the student policy. We train with the ADAM optimizer

to minimize MSE loss. We run the optimization process for 10M steps, training on data collected over the last 1M steps. Training the adaptation module takes approximately 20 minutes. Both networks are trained with the deep learning framework PyTorch. For more efficient inference and resource allocation on the onboard computer, we use Mobile Neural Network (MNN) [40], [41] to convert trained models to MNN formats to optimize their inference speed and overhead. Table III presents the average inference time measured on the RB5 platform for onboard computation. To meet the required 500Hz control frequency, the combined inference time of the controller and adaptation module must be less than or equal to 2ms. MNN satisfies this requirement with a total average inference time of 0.165ms, whereas PyTorch exceeds the limit with a total of 78.850ms, making real-time onboard inference infeasible.

TABLE III: The average inference time for the PyTorch and MNN frameworks measured over a 10-second window on the Qualcomm Robotics RB5 platform.

	Mean Inference Time (ms)			
	PyTorch		MNN	
Low-level Controller ()	73.626	2.692	0.078	0.003
Adaptation Module ()	5.224	0.708	0.087	0.005

IV. SIMULATION EXPERIMENTS

In this section, we evaluate the performance of our controller through multiple simulation experiments. We begin by establishing a set of baseline methods and justifying their selection. Subsequently, we evaluate each method and ours on the task of trajectory tracking for randomized quadcopters. The results of these initial tests motivate us to further challenge our approach on quadcopters that significantly deviate from the training distribution. The simulation experiments offer a controlled environment to assess our approach on aspects of robustness, adaptivity and generalization, thus paving the way for subsequent hardware experiments.

A. Baselines Setup

We compare our approach with a set of baselines in the simulation. The task is to evaluate the tracking performance of a randomly sampled quadcopter along random trajectories. We randomize quadcopters according to our design-informed domain randomization technique outlined in Section E. The testing range is listed in Table II and a sample of typical desired trajectories is shown in Figure 4. We choose a nominal quadcopter model m_{norm} , which is obtained by setting $\sigma = 0.5$ when sampling without noise added.

We choose several different sets of high-level and low-level controllers as baselines from prior work. We include the implementation details of all baselines in Appendix A. Most baseline names follow the format high-level controller-low-level controller, with exceptions explicitly noted. First, we include PID-PD and PID-PD_n as reference baselines. PID-PD serves as an upper performance bound, as its low-level controller has access to the ground-truth model parameters of each sampled quadcopter. This makes it equivalent to the exper-

imental controller used during training, as described in Section D. In contrast, PID-PD_n serves as a lower performance bound, since its low-level PD controller only uses nominal parameters from the model m_{norm} and does not adapt to individual quadcopters. These two baselines provide a sanity check: any effective adaptation method should achieve performance within this range. To ensure that the adaptation task is handled entirely by the low-level controller, all high-level controllers in both the baselines and our framework use the same control parameters across all tested vehicles. These control gains are tuned specifically to optimize performance on the nominal model m_{norm} . Next, we include L₁-PD_n and PID-L₁ as additional baselines for evaluating our method. L₁-PD_n employs the L₁ adaptive high-level controller ([4], [5]) to compensate for model uncertainties while keeping the low-level PD controller fixed. In contrast, PID-L₁ applies L₁ as a low-level adaptive controller. These baselines serve a dual purpose: they act as additional benchmarks for evaluating our approach while also validating our core assumption about adaptation. We design our method as a low-level controller under the assumption that adaptation across the diverse parametric range in Table II is more effective when handled at this control level. In other words, an adaptive high-level controller alone cannot sufficiently compensate for the model disparity in our problem, which involves controlling quadcopters with significant differences in design and actuators. The L₁ baselines allow us to test this assumption by comparing their performance when adaptation occurs at different control levels. Finally, we compare our method to state-of-the-art adaptive controllers. We include Geo-A, a geometric adaptive controller that operates at both high and low levels, so is an exception in our naming convention. We also include PD-INDI-A, which combines a PID high-level controller with a low-level adaptive INDI controller [13]. These baselines provide further context on how our method compares to established nonlinear adaptive control techniques.

At the beginning of each experiment, the quadcopter is spawned with a hovering state. The trajectory to track is generated with the motion primitive generation algorithm [6]. At the end condition sampled from the test range in Table II, the experiment is considered successful if the position tracking error is within 2m at every point of the trajectory.

B. Results

The results of the simulation experiments are reported in Table IV. We compare the five approaches under three metrics: (i) the success rate, (ii) the maximum position tracking error, (iii) the root-mean-square error (RMSE) in position tracking, and (iv) the RMSE in velocity tracking. We rank the methods according to the success rate and the tracking performance. Given the very large amount of quadcopter variations, PID-INDI achieved the lowest success rate and the largest tracking error. In contrast, PID-PD has a 100% success rate with the lowest tracking error, since it uses the ground-truth parameters of the quadcopter in computing the control inputs. Without access to the ground truth

Fig. 4: Visualization of typical desired quadcopter trajectories in 3D space during simulated tests. The color gradient represents the magnitude of the velocity at each point along the trajectory. We sample 50 trajectories from the origin with the distribution defined by Table II. The initial conditions of all trajectories are hovering at origin.

TABLE IV: We choose 6 baselines: PID-PD, PID-PD_n, L₁-PD_n, PID-L₁, Geo-A and PID-INDI-A. The PID-PD has access to all ground-truth system parameters and thus could be regarded as the expert. We compare their performance on the task of tracking random quadcopters along trajectories. The test ranges are defined in Table II. The metrics are the success rate, the maximum position tracking error, the position and velocity RMSE between the actual quadcopter trajectory and the reference trajectory. The results are from 100 experiment for each baseline.

	Success Rate	Max Pos. Err (m)	Position RMSE (m)	Velocity RMSE (m/s)
PID-PD _n	22%	1.565	0.510 0.372	0.845 1.066
L ₁ -PD _n	62%	1.105	0.186 0.167	0.278 0.392
Geo-A	64%	1.033	0.128 0.160	0.174 0.263
PID-INDI-A	67%	0.333	0.063 0.060	0.075 0.082
PID-L ₁	77%	1.304	0.221 0.242	0.357 0.481
PID-Ours	100%	0.311	0.148 0.075	0.129 0.066
PID-PD (Expert)	100%	0.221	0.061 0.057	0.059 0.050

parameters as PID-PD but with adaptation to the unknown dynamics, the flight performance of ℓ_1 controllers significantly increase. However, with adaptation at high-level, the PID_n achieves a lower success rate than its counterpart. PID-L₁ with adaptation at low-level. Since tracking errors are only computed in successful runs, the PID_n achieves a slightly lower tracking error. This result has shown that an adaptive low-level controller tends to perform better with the large model disparity across the platforms, which justifies our assumption for our controller design.

The two nonlinear adaptive baselines Geo-A and PID-INDI-A both show better tracking performance than ℓ_1 controllers. Specifically, PID-INDI-A achieves the lowest tracking error wider than the training set. Figure 5 provides a visualization in successful flights among all methods. However, both baselines achieve a similar success rate of approximately 65%. Specifically, we often observe them failing near the boundaries of the adaptation range, where the dynamics deviate significantly from the nominal model. Moreover, their performance relies on prior knowledge of the reference model θ_{norm} , and INDI-A in particular also requires access to current

Fig. 5: Visualization of the differences between the training set and the $\theta = 8$ testing set. The scatter plot shows 2,000 randomly sampled quadcopters based on arm length, mass, and torque-to-thrust coefficient C_T . The testing set exhibits a significantly wider distribution than the training set.

motor speed measurements and accurate estimates of angular acceleration and torque, whereas our method operates without such information. Despite this, our method achieves a 100% success rate and the lowest maximum position error among all baselines, with only a slightly higher average tracking error than the expert controller with access to the true parameters.

C. Generalization

We evaluate the task of tracking trajectories on held-out quadrotor parameter range. In particular, we aim to determine the extent to which deviations from the nominal model cause our controller and other baseline controllers to fail. We use the same baselines as in previous sections, with the nominal model θ_{norm} now obtained at the mid-point of the training range. For ease of representation, we express θ_{norm} in a numeric way, with its value equal to the scaling constant. Therefore, $\theta_{norm} = 0.5$ and $\theta \in [0, 1]$ is the training set of Table II.

We use the metrics

$$J = \max_j \theta_{norm}^j \quad (18)$$

to define the extent of the range of sampled quadrotor parameters. In particular, when $\theta = 0$, the sampled quadrotor is the nominal model θ_{norm} with noises; when $\theta = 0.5$, $\theta \in [0, 1]$ is the training set in Table II. We extend up to 8 to evaluate the task of trajectory tracking for randomly sampled quadrotors, in which the sampling range is 16 times wider than the training set. Figure 5 provides a visualization of the differences between the two sets. We randomly sample 2,000 quadcopters within the $\theta = 8$ range and within the training set, plotting them as scatter points based on arm length, mass, and torque-to-thrust coefficient. This illustration highlights the differences in size, mass, and motor strength between the training and testing sets. It is evident that the testing set is significantly outside the training distribution.

Fig. 7: To evaluate the sim-to-real gap, we control the large quadrotor with our proposed controller along circular trajectories at 3 speed settings: Slow, Medium and Fast. The figure illustrates the error bars for the distribution of position and velocity tracking errors. Both plots exhibit similar trends and magnitudes, suggesting that our simulator effectively reflects real-world dynamics and thus supports the reliability of our results. These results are derived from 10 simulated flights and 3 real-world flights.

V. HARDWARE EXPERIMENTS

In this section, we transition from simulation to hardware experiments. We first investigate the sim-to-real correlation to ensure the validity of our simulation results. Subsequently, we conduct a comparative analysis on disturbance rejection tasks, comparing our method against the best baseline identified in our simulation tests.

A. Sim-to-Real Correlation

Fig. 6: We evaluate the performance of our method and all baselines on extended quadrotor parameters range unseen at training time. We use metrics $\epsilon = \max_j \epsilon_{\text{norm}, j}$, the maximum difference of all sampled quadcopters away from the nominal model, to define the quadrotor randomization range. We plot: the success rate (Top), the box plot of the position tracking error (Middle), and the box plot of the position tracking error of our method and all baselines over the parameter randomization range. At each data point, the result is calculated over 100 experiments. All sampled quadcopters within the shaded area between 0 and 0.5 belong to the training range (Table II). Note that for better visualization, the x-axis is not to scale.

The success rate and the position and velocity tracking error distribution are reported in Figure 6. Our method achieves a near 100% success rate until $\epsilon = 8$ where it drops to 95%. In contrast, all other baselines, except PID-PD, exhibit significant performance degradation as the model mismatch from the nominal model increases. In addition, the average position tracking error at $\epsilon = 8$ of our method is still close to that at $\epsilon = 0$, with only 37.0% increase. Compared to the strongest baseline PID-L₁ in terms of success rate in Section IV-B whose position tracking error has grown by 483.7% compared to that at the nominal model.

We validate our simulation results through two sets of hardware and simulation experiments to examine the simulation-to-reality gap. We fly the large quadrotor along circular trajectories with our control framework RID-Ours at 3 speed settings: Slow, Medium and Fast. Subsequently, we simulate the same flight paths of the same vehicle with our method again using the Flightmare simulator [37]. The trajectories involve circling with a 1-meter radius with completion in different durations: 8s (Slow), 4s (Medium), and 3s (Fast). The results, illustrated in Figure 7, show the distribution of position and velocity tracking errors for both sets of experiments across all speed settings. We also compute the Pearson Correlation Coefficient [42], [43], which shows a moderate positive correlation (0.652) between the position tracking errors in the simulation and real-world tests, with a statistically significant P-value of 0.002. These findings suggest that our simulator captures the dynamics of the real world reasonably well, supporting the reliability of our simulation results.

B. Baseline Comparison

We test our approach in the physical world and compare its performance to the PID-PD controller that has access

Fig. 8: (a) Large quadrotor and small quadrotor mounted with an off-center payload. For the large vehicle, we mount a 200g payload to the farthest end of the body frame from the center of gravity. For the small one, we mount a 30g payload directly under one of its motors. (b) Both quadcopters experiencing a 20% thrust loss from one of its actuator, which is achieved by hard-coding the firmware code. (c) Small quadrotor tracking a circular trajectory under wind up to 3.5m/s. Large quadrotor undergoes this experiment with the same setup.

training. Our controller is robust to wind disturbances with a comparable tracking performance as **PID-PD** on the large quadrotor and a significantly smaller tracking error on the small quadrotor. The small size and weight of this platform make it more susceptible to the interaction of the wind with its body and rotors, which can alter its aerodynamic properties, such as affecting the effective angle of attack on the rotors. Therefore, the model mismatch in terms of alteration in aerodynamic properties is more prominent on the small quadrotor than on the large quadrotor. Our controller can adapt well to such disturbances. Across all sets of experiments, position and velocity errors correlate with high-level tracking command errors, as larger high-level errors lead to compounded position and velocity inaccuracies. This further emphasizes the importance of robustness and adaptivity of the low-level controller.

C. Adaptation Analysis

We consider the controller performance for an off-center payload, using the large quadcopter. The result is shown in Figure 9. During takeoff, all components exhibit

Fig. 9: Visualization of the off-center payload experiment on the large quadrotor. The plots show Top: motor speeds commanded by our low-level controller, Middle: absolute position tracking throughout the experiment, and Bottom: each element of the estimated 8-dimensional intrinsic \mathbf{z}_t . The timeline spans from takeoff to approximately 10 seconds after the payload is added. Shaded regions indicate phase transitions, with annotations indicating key events. In the bottom plot, elements of \mathbf{z}_t that exhibit more than a 10% change in their average value before and after payload attachment are highlighted, showing the components most affected by the payload disturbance.

variations as the quadrotor stabilizes. The vehicle transitions from stationary on the ground to takeoff at the commanded velocity within approximately 0.2 seconds, aligning with the adaptation module's time window (100 state-action pairs at 500Hz). This adaptation process is significantly faster than approaches that rely on online system identification [45], which typically require 10–15 seconds of flight to converge to an accurate model estimate.

When the payload is added, multiple components undergo a sharp transient response, indicating an update in the controller's internal representation. The highlighted components show a sustained shift, suggesting that these dimensions capture key physical properties such as mass distribution and inertial variations. In contrast, the average values of z_2 and z_6 remains largely unchanged after convergence, suggesting that these dimensions encode properties that are unaffected by an off-center payload, such as lateral forces or yaw torque.

Fig. 10: Comparison of episode rewards (top) and episode lengths (bottom) for policies trained using both IL and RL, versus RL only, across 10 consecutive random seeds (with the bold line representing the mean and the shaded area representing the 95% confidence interval). The episode length is normalized by dividing the vehicle's surviving duration by the maximum episode duration. An episode length of 1 indicates that the vehicle controlled by the trained policy survives for the entire episode without crashing. For better visualization, we plot the reward curve in log scale.

TABLE VI: Comparison of average episode reward and length with 95% confidence interval (CI) over the last 10% of the training process for IL+RL and RL-only policies.

	Mean Episode Reward 95% CI	Mean Episode Length 95% CI
IL+RL	-31.72 22.02	0.97 0.05
RL	-102.55 76.39	0.79 0.20

VI. ABLATION STUDY

Training a low-level controller that adapts across different quadrotors involves various complexities. This section presents an ablation study analyzing the training curves to evaluate the impact of the IL component and reward design. Additionally, we examine how the RL and IL components, and our control level selection, influence performance.

A. Training Curve Analysis

1) Integration of IL: We adopt a dual strategy which combines IL and RL and adaptively adjust the relative weight

Fig. 11: Comparison of episode reward for policies trained with torque tracking, versus with angular velocity tracking, across 10 consecutive random seeds (with the bold line representing the mean and the shaded area representing the 95% confidence interval). The episode reward does not include the torque/angular velocity reward term as they are the variation in reward design.

TABLE VII: Comparison of average episode reward with 95% confidence interval (CI) over the last 10% of the training process for policies with torque tracking and with angular velocity tracking. The episode reward does not include the torque or angular velocity tracking reward for comparability.

	Mean Episode Reward 95% CI
Torque Tracking	-3.91 4.67
Angular Velocity Tracking	-8.74 8.15

between the two losses. This learning approach improves the training of the low-level controller compared to training solely with RL, as in our previous conference paper [20].

Figure 10 presents an ablation of the IL component of the loss. We also report the average episode reward and length with 95% confidence interval (CI) over the last 10% of training process in Table VI. The policy trained with both IL and RL exhibits steady improvement in both reward and episode length throughout the learning phase, maintaining an episode length close to 1 after approximately 10M steps. In contrast, the RL-only baseline shows less consistent performance with a larger variance. Although its episode length approaches 1 around 20M steps, it soon begins to diverge, indicating instability. Additionally, the RL-only baseline maintains a consistently lower average reward compared to the IL-and-RL method, highlighting inferior tracking performance. As training progresses, the instability worsens, leading to a decline in reward. To further analyze these components, we conduct an ablation study in the context of our simulation experiments and evaluate the roles of the RL and IL components in Section B.

2) Reward Design: We reward torque tracking instead of angular velocity tracking, as described in Section B. While angular velocity is a high-level command and a key observation, our experiments show that torque tracking leads to better sim-

To support this, we compare policies trained with torque

Fig. 12: We perform an ablation study under the same experimental conditions as Figure 6, comparing our method(s) against three variations: (i) RL-Only, which removes the IL component and trains solely via reinforcement learning, (ii) IL-Only, which trains the controller exclusively through imitation learning using DAGger [46] and (iii) End-to-End a single learned policy with our method but integrating high-level and low-level controllers. The results are taken directly from Figure 6. The results are shown Top: success rate, and Bottom: the position tracking error (box plots). At each data point, the result is calculated over 100 experiments. All sampled quadcopters within the gray shaded area belong to the training range of Table II. Note that for better visualization, the x-axis is not to scale.

tracking and angular velocity tracking in Figure 11, keeping all other reward components and hyperparameters identical. Table VII reports the average reward over the last 10% episode length, thereby improving success rates. This explains why RL-Only outperforms IL-Only in terms of success rate. When combined, RL can prioritize success rate optimization, as the IL loss naturally handles tracking error minimization. This complementary relationship allows the two methods to reinforce each other, resulting in improved overall performance. Future work could explore this hypothesis further, investigating the underlying mechanisms and conditions that enable a synergy between IL and RL.

B. Simulation Performance

In this section, we present an ablation study to investigate the contribution of individual components in our method design among all tested variations. In contrast, the other variations (within the context of our simulation experiments. Specifically, we compare our method against three variations: RL-Only, IL-Only and Ours) all utilize a low-level controller. The failure of End-to-End learning likely comes from the challenge of jointly encoding both high-level trajectory tracking and low-

on reinforcement learning for training, (ii) IL-Only, which trains the controller solely through imitation learning using DAGger [46] without separation of networks and phases as in our framework, and (iii) End-to-End which replaces our hierarchical control structure with a unified policy that directly fuses high-level and low-level control, trained using our method. The first two variants provide insight into the significance of individual training components, extending the analysis from the training curves in Section IV-A. The third variation focuses on justifying our design choice for separate control levels. We perform the experiment on the same generalization task described in Section IV-C. The success rate and the distribution of position tracking errors are presented in Figure 12. Note that the results for Ours are the same as shown in Figure 6.

1) RL and IL Ablation: Both RL-Only and IL-Only experience a decline in success rate as the model mismatch increases beyond the training set, ultimately dropping to around 60%. RL-Only demonstrates a slightly higher success rate than IL-Only. In terms of tracking error, IL-Only achieves the smallest distribution among the methods. However, both approaches exhibit increased tracking error as the model mismatch grows. At $\epsilon = 8$, the average tracking error for IL-Only increases by 92.4%, while for RL-Only, it grows by 61.4%, compared to their respective values at $\epsilon = 0$. In contrast, our method, which combines RL and IL, maintains a near 100% success rate across all levels of model mismatch while achieving consistent tracking performance. This raises the question: how can the combination of two methods, each with limited generalization capability, yield better results?

We hypothesize that it is because IL and RL optimize distinct objectives. IL focuses on minimizing instantaneous tracking error, while RL targets both tracking error and success rate. RL achieves this dual optimization by incorporating a reward signal for tracking error, as described in Section IV-C, and utilizing episode termination conditions to influence success rate. As a result, RL may accept higher tracking errors to extend episode length, thereby improving success rates. This explains why RL-Only outperforms IL-Only in terms of success rate. When combined, RL can prioritize success rate optimization, as the IL loss naturally handles tracking error minimization. This complementary relationship allows the two methods to reinforce each other, resulting in improved overall performance. Future work could explore this hypothesis further, investigating the underlying mechanisms and conditions that enable a synergy between IL and RL.

2) End-to-End Evaluation: End-to-End is trained using our proposed method but as a unified policy that combines high-level and low-level control. Its input includes the same features as our policy, along with additional information: position (\mathbf{R}^3), velocity (\mathbf{R}^3), and the difference between the current and goal positions (\mathbf{R}^3). Its output, like our policy, is motor speed, which motivates the name End-to-End. Despite being trained with the same method and without ablation, End-to-End demonstrates

the lowest success rate and the largest average tracking error among all tested variations. In contrast, the other variations (RL-Only, IL-Only and Ours) all utilize a low-level controller. The failure of End-to-End learning likely comes from the challenge of jointly encoding both high-level trajectory tracking and low-

level motor adaptation within a single policy. Maintaining a modular structure allows each level to focus on distinct aspects of control, facilitating generalization beyond the training set.

Moreover, when compared to state-of-the-art model-based methods in Figure 6, its performance is similarly poor. At $n = 8$, End-to-End achieves a success rate of only 28%, which is marginally better than $PID-PD_n$ and L_1-PD_n , both of which lack adaptation in the low-level controller. This comparison in parallel also highlights the importance of separating control levels and employing an adaptive low-level controller to handle the large model disparities defined in our problem.

VII. CONCLUSION

This work demonstrates how a single adaptive controller can effectively bridge the gap between high-level planning and the intricate physical dynamics by adapting to model disparities between quadcopters down to the motor level. Our design focuses on creating a low-level controller intended to replace traditional low-level quadcopter controllers, thereby eliminating the need for accurate model estimation and iterative parameter tuning. Our approach leverages a combination of imitation learning from model-based controllers and reinforcement learning to address the challenges of training a sensor-actuator controller at high frequencies. The introduction of an instant reward feedback ensures that the controller remains responsive and agile. In addition, we develop a quadcopter randomization method during training that aligns with real-world constraints, further enhancing its adaptability. The controller's ability to estimate a latent representation of system parameters from sensor-action history, along with realistic domain randomization, empowers it to generalize across a broad spectrum of quadcopter dynamics. This capability extends to unseen parameters, with an adaptation range up to 16 times broader than the training set. The single policy trained solely in simulation can be deployed zero-shot to real-world quadcopters with vastly different designs and hardware characteristics. It also demonstrates rapid adaptation to unknown disturbances such as off-center payloads, wind, and loss of efficiency in motors. These results highlight the potential of our approach for extreme adaptation for drones and other robotic systems, while enabling robust control in the face of real-world uncertainties.

VIII. ACKNOWLEDGEMENT

This work was supported by the Hong Kong Center for Logistics Robotics, the DARPA Transfer from Imprecise and Abstract Models to Autonomous Technologies (TIAMAT) program, the Graduate Division Block Grant of the Dept. of Mechanical Engineering, UC Berkeley, and ONR MURI award N00014-21-1-2801. The authors would like to thank Ruiqi Zhang and Teaya Yang for their help with the experiments. The authors also would like to thank Daniel Gehrig for his valuable input in the ablation analysis of our method. The experimental testbed at the HiPeRLab is the result of the contributions of many people, a full list of which can be found at hyperlab.berkeley.edu/members/.

Fig. 13: The control diagram of $PID-L_1$.

APPENDIX A BASELINE IMPLEMENTATION

A detailed explanation of each baseline's implementation is provided to support reproducibility. The baselines are structured into high-level and low-level controllers. Most baseline names follow the format high-level controller/low-level controller, with exceptions explicitly noted. For instance, $PID-PD$ indicates that PID serves as the high-level controller, while PD represents the low-level controller. Identical control names imply the same implementation, and explanations are omitted if they were already described for previous baselines.

A. $PID-PD$

1) PID : The high-level PID controller is implemented as described in Section III. It generates high-level commands, including body rates and mass-normalized collective thrust, for other low-level controllers to track.

2) PD : This baseline matches the expert controller used during training, as outlined in Section IV-D. Unlike other baselines, it has access to ground-truth model parameters for control calculations. Consequently, it serves as both an expert reference and a performance upper bound, providing context for evaluation.

B. $PID-PD_n$

1) PD_n : This baseline is similar to PD but calculates control using only the nominal model m_{norm} . As such, it represents a performance lower bound for comparison.

C. $PID-L_1$

1) L_1 at Low Level: We implement the L_1 adaptive controller as an augmentation to the $PID-PD_n$ control, applying the adaptive controller at the low level. Beyond serving as a baseline, it also acts as a counterpart to PD_n , which applies the same adaptation within the high-level control structure. The comparison between the two methods can validate our design choice of control hierarchy. The implementation is same as [10]. The controller's block diagram is shown in Figure 13. The L_1 control receives the desired motor thrust from PD_n and computes the augmented thrust using the control and adaptation law defined in (6-13) in [10]. The resulting thrust command is then converted into motor speed. Unless otherwise specified, all model constants used for control calculations in this framework and in all baselines, except PD are derived from the nominal model.

Fig. 14: The control diagram of L_1 -PD_n

Fig. 15: The control diagram of Geo-A

D. L_1 -PD_n

1) L_1 at High Level: We implement a similar L_1 controller as described previously but the uncertainty is compensated at the level of thrust and body torque. We refer to the framework in [47] in which the L_1 controller gives thrust and torque command. The control and adaptation law is defined in (19-24) in [47]. Figure 14 outlines the block diagram of the controller.

E. Geo-A

This baseline is a geometric adaptive controller both at the high level and low level, abbreviated as Geo-A, an exception to our naming convention. We choose Geo-A as a baseline to contextualize our proposed approach. This method ensures precise trajectory tracking and robust stabilization for quadrotors by adapting to uncertainties and disturbances in a geometrically consistent manner.

We implement the geometric adaptive controller based on [12], replacing the original geometric attitude controller with a tilt-prioritized control from [48] to enable tracking of dynamically infeasible trajectories. Figure 15 provides an overview. The position controller, based on (23-25) in [12], computes target acceleration, which is decomposed into target thrust and target attitude using (14-17) in [48]. The attitude is processed by tilt-prioritized control ((25-28) in [48]), and the resulting angular velocity and thrust are sent to the angular velocity controller ((15-16) in [12]). Finally, thrust commands are mapped to motor speeds.

F. PID-INDI-A

1) INDI-A: We implement the adaptive INDI controller based on [13] and the block diagram is detailed in Figure 16. INDI is a sensor-based control which uses instant sensor measurement to represent the system dynamics so that to be more robust to unmodelled uncertainties in the rotation

Fig. 16: The control diagram of PID-INDI-A.

dynamics. To enhance its adaptiveness [49] introduces onboard adaptive parameter estimation to update control effectiveness.

The original implementation requires measurements of angular acceleration, motor speed, and motor speed acceleration. The setup provides more informative observations than other baselines and our method, which do not have access to motor feedback. For a fair comparison, we modify the control law to reduce reliance on motor feedback, using only motor speed estimates obtained through a first-order response model with a time constant of 10ms while preserving the original linear update adaptation law.

After knowing the desired mass-normalized thrust and angular velocity from the high-level controller, we can compute the desired torque using (6). Then we can compute the desired motor speed command using (7) and (8), yielding:

$$m \mathbf{c}_{des}^{c;des} = M^{-1} \mathbf{F}_{des} \quad (19)$$

$$= M^{-1} \mathbf{C}_F \mathbf{a}_{des}^2 \quad (20)$$

We want to use instantaneous sensor feedback to compute the desired motor speed to be against model uncertainty and disturbance. Therefore, we decompose the thrust and torque into current measurement and desired value (see (30) and (31) of [49] for detailed derivations of torque decomposition).

$$\mathbf{c}_{des}^c = \mathbf{c} + (\mathbf{c}_{des}^c - \mathbf{c}) \quad (21)$$

$$\mathbf{c}_{des}^t = \mathbf{c} + \mathbf{J}(\mathbf{l}_{des} - \mathbf{l}) \quad (22)$$

So that the effect of unmodeled rotational dynamics is captured by the measurement of angular acceleration and torque. Note the desired angular acceleration is gained through (5). Similarly, the desired motor speed is decomposed as:

$$m \mathbf{c}_{des}^{c;des} = M^{-1} \mathbf{C}_F \mathbf{a}^2 + M^{-1} \mathbf{C}_F (\mathbf{a}_{des}^2 - \mathbf{a}^2) \quad (23)$$

By rearranging the terms, we obtain the following equation to solve for \mathbf{a}_{des}^2 :

$$\mathbf{a}_{des}^2 - \mathbf{a}^2 = \mathbf{C}_F^{-1} M^{-1} \mathbf{J} (\mathbf{c}_{des}^c - \mathbf{c}) \quad (24)$$

This equation can then be reformulated into the INDI framework by introducing the control allocation matrices \mathbf{G}_1 and \mathbf{G}_2 :

$$\mathbf{a}_{des}^2 - \mathbf{a}^2 = \mathbf{G}_1 (\mathbf{c}_{des}^c - \mathbf{c}) + \mathbf{G}_2 (\mathbf{l}_{des} - \mathbf{l}) \quad (25)$$

with the adaptation law defined as

$$G(k) = G(k-1) - \frac{C}{L} (a^2) \frac{C^T}{L} \quad (26)$$

$$G = G_1 - G_2 \quad (27)$$

where Δ denotes the difference between the current and previous sampled variables. A more detailed analysis on the stability and performance of this method could be found in [16].

REFERENCES

- [1] E.-H. Zheng, J.-J. Xiong, and J.-L. Luo, "Second order sliding mode control for a quadrotor uav," *JSA Transactions*, vol. 53, no. 4, pp. 1350–1356, 2014, disturbance Estimation and Mitigation.
- [2] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.
- [3] I. M. Mareels, B. D. Anderson, R. R. Bitmead, M. Bodson, and S. S. Sastry, "Revisiting the mit rule for adaptive control," *IFAC Proceedings Volumes* vol. 20, no. 2, pp. 161–166, 1987.
- [4] C. Cao and N. Hovakimyan, "Design and analysis of a novel l1 adaptive control architecture with guaranteed transient performance," *IEEE Transactions on Automatic Control*, vol. 53, no. 2, pp. 586–591, 2008.
- [5] N. Hovakimyan and C. Cao, *l1 adaptive control theory: Guaranteed robustness with fast adaptation* SIAM, 2010.
- [6] P. A. Ioannou, A. M. Annaswamy, K. S. Narendra, S. Jafari, L. Rudolph, R. Ortega, and J. Boskovic, "l1-adaptive control: Stability, robustness, and interpretations," *IEEE Transactions on Automatic Control*, vol. 59, no. 11, pp. 3075–3080, 2014.
- [7] R. Ortega and E. Panteley, "L1-adaptive control always converges to a linear pi control and does not perform better than the IFAC Proceedings Volumes," vol. 47, no. 3, pp. 6926–6928, 2014, 19th IFAC World Congress.
- [8] —, "Adaptation is unnecessary in l1-adaptive control: What makes an adaptive controller adaptive?" *IEEE Control Systems Magazine* vol. 36, no. 1, pp. 47–52, 2016.
- [9] R. Tao, P. Zhao, I. Kolmanovsky, and N. Hovakimyan, "Robust adaptive mpc using uncertainty compensation," *2024 American Control Conference (ACC) 2024*, pp. 1873–1878.
- [10] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 690–697, 2022.
- [11] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "L1-adaptive mppi architecture for robust and agile control of multirotors," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2020*, pp. 7661–7666.
- [12] F. A. Goodarzi, D. Lee, and T. Lee, "Geometric adaptive tracking control of a quadrotor unmanned aerial vehicle on se(3) for agile maneuvers," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 9, Jun. 2015.
- [13] E. Smeur, Q. Chu, and G. Croon, "Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 39, pp. 1–12, 12 2015.
- [14] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters* vol. 2, no. 4, p. 2096–2103, Oct. 2017.
- [15] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, 2019.
- [16] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2021*, pp. 1205–1212.
- [17] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," *2022 International Conference on Robotics and Automation (ICRA) 2022*, pp. 10504–10510.
- [18] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.
- [19] A. Tagliabue, D.-K. Kim, M. Everett, and J. P. How, "Demonstration of efficient guided policy search via imitation of robust tube mpc," *2022 International Conference on Robotics and Automation (ICRA) 2022*, pp. 462–468.
- [20] T. Zhao, A. Tagliabue, and J. P. How, "Efficient deep learning of robust adaptive policies using tube mpc-guided data augmentation," *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2023*, pp. 1705–1712.
- [21] A. Tagliabue and J. P. How, "Efficient deep learning of robust policies from mpc using imitation and tube-guided data augmentation," *IEEE Transactions on Robotics*, vol. 40, pp. 4301–4321, 2024.
- [22] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural-ly enables rapid learning for agile flight in strong winds," *Science Robotics*, vol. 7, no. 66, p. eabm6597, 2022.
- [23] K. Huang, R. Rana, A. Spitzer, G. Shi, and B. Boots, "Datt: Deep adaptive trajectory tracking for quadrotor control," *Proceedings of The 7th Conference on Robot Learning*, ser. *Proceedings of Machine Learning Research*, J. Tan, M. Toussaint, and K. Darvish, Eds., vol. 229. PMLR, 06–09 Nov 2023, pp. 326–340. [Online]. Available: <https://proceedings.mlr.press/v229/huang23a.html>
- [24] G. Feng, H. Zhang, Z. Li, X. B. Peng, B. Basireddy, L. Yue, Z. SONG, L. Yang, Y. Liu, K. Sreenath, and S. Levine, "Genloco: Generalized locomotion controllers for quadrupedal robots," *Proceedings of The 6th Conference on Robot Learning*, ser. *Proceedings of Machine Learning Research*, K. Liu, D. Kulis, and J. Ichnowski, Eds., vol. 205. PMLR, 14–18 Dec 2023, pp. 1893–1903. [Online]. Available: <https://proceedings.mlr.press/v205/feng23a.html>
- [25] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," *Proceedings of Robotics: Science and Systems*, Corvallis, Oregon, USA, July 2020.
- [26] J. Eschmann, D. Albani, and G. Loianno, "Learning to fly in seconds," *IEEE Robotics and Automation Letters*, pp. 1–8, 2024.
- [27] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2019*, pp. 59–66.
- [28] P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt, "Learning to fly via deep model-based reinforcement learning," *arXiv preprint arXiv:2003.08876* 2020.
- [29] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller, "Learning a single near-hover position controller for vastly different quadcopters," in *2023 IEEE International Conference on Robotics and Automation (ICRA) 2023*, pp. 1263–1269.
- [30] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *RSS: Robotics Science and Systems* 2021.
- [31] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics* vol. 5, no. 47, p. eabc5986, 2020.
- [32] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotors," *IEEE Robotics & Automation Magazine* vol. 19, no. 3, pp. 20–32, 2012.
- [33] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadcopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.
- [34] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. van de Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Proceedings of the Conference on Robot Learning*, ser. *Proceedings of Machine Learning Research*, L. P. Kaelbling, D. Kravic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 317–329. [Online]. Available: <https://proceedings.mlr.press/v100/xie20a.html>
- [35] H. Huang, A. Loquercio, A. Kumar, N. Thakkar, K. Goldberg, and J. Malik, "Manipulator as a tail: Promoting dynamic stability for legged locomotion," in *2024 IEEE International Conference on Robotics and Automation (ICRA) 2024*, pp. 9712–9719.
- [36] A. Loquercio, A. Kumar, and J. Malik, "Learning visual locomotion with cross-modal supervision," *2023 IEEE International Conference on Robotics and Automation (ICRA) 2023*, pp. 7295–7302.
- [37] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Proceedings of the 2020 Conference on Robot Learning*, ser. *Proceedings of Machine Learning Research*, J. Kober, F. Ramos, and C. Tomlin, Eds., vol. 155. PMLR, 16–18 Nov 2021, pp. 1147–1157. [Online]. Available: <https://proceedings.mlr.press/v155/song21a.html>
- [38] M. W. Mueller, "Multicopter attitude control for recovery from large disturbances," *arXiv preprint arXiv:1802.09143* 2018.

