A Learning-based Quadcopter Controller with Extreme Adaptation

Dingqi Zhang[1], Antonio Loquercio[2], Jerry Tang[1], Ting-Hao Wang[1],
Jitendra Malik[3], and Mark W. Mueller[1]

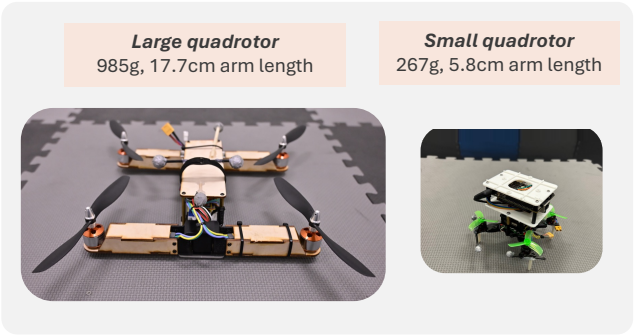
***Abstract*— This paper introduces a learning-based low-level controller for quadcopters, which adaptively controls quadcopters with significant variations in mass, size, and actuator capabilities. Our approach leverages a combination of imitation learning and reinforcement learning, creating a fast-adapting 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

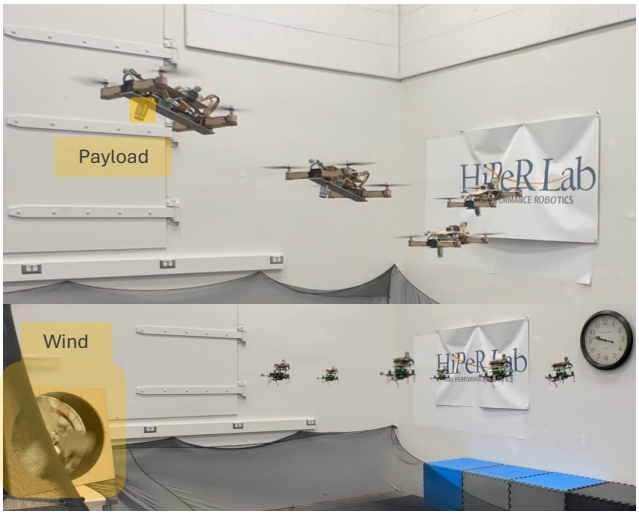
The agile nature of quadcopters and the necessity for precise control in dynamic environments create a unique context for exploring control strategies. Model-based controllers for quadcopters generally rely on estimates of the vehicle's properties, including inertia, motor constants, and other parameters. 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, motor failures, and wind.

The authors are with the [1]High 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, wtyngh, malik, mwm}@berkeley.edu



(a) Hardware Details



(b) Deployment

Fig. 1: Our adaptive controller can control quadcopters with vast difference including but not limited to mass, arm length, and actuators while also show disturbance rejection. **(a)** Overview of our hardware setup consisting of two quadcopters with mass differing by a factor of 3.68, arm length by 3.1 and totally different propellers. **(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.

A. Related Work

The development of fixed-parameter controllers, while foundational, is inherently limited by their lack of real-time 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. Recent work has attempted to overcome these limitations by combining $\mathcal{L}_1$ adaptive control with nonlinear online optimization schemes, such as model predictive control (MPC) [6–8]. These methods have achieved impressive results, but still require explicit and accurate knowledge of the reference model, which is crucial for adaptation.

Recent advances in data-driven control strategies have shown promising results for quadcopter stabilization [9, 10], or waypoint tracking flight [11, 12], as well as agile racing against human pilots [13]. Model-free reinforcement learning in [13] 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 MPC through imitation learning [14, 15]. Another approach is augmenting the learned controller with classical adaptive control designs during deployment for fast disturbance estimation and online adaptation [16, 17]. 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 [18], 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 [9, 12, 19–22], 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 ($\approx$ 20%) around a nominal model, 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 [23] and introduce three key technical innovations: (1) a dual training strategy that combines behavior cloning 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 reward and 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 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. For clarity, we provide a list of symbols and notation used throughout the paper in Table I.

A. Control Structure

Cascade control systems are instrumental in managing complex dynamic systems by decomposing them into a hierarchy of simpler nested subsystems. In this structure, 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.

TABLE I: List of Symbols

Notation			
x	Scalar quantity	$\boldsymbol{x}$	Vector quantity (e.g., $\boldsymbol{\omega}$)
x_{des}	Commanded quantity	$\hat{x}$	Estimated quantity
$x_{\min}$	Minimum value	$x_{\max}$	Maximum value
State Variables			
$\boldsymbol{p}$	Position	$\boldsymbol{v}$	Velocity
$\boldsymbol{q}$	Attitude (quaternion)	ψ	Yaw angle
$\boldsymbol{\omega}$	Angular velocity	$\boldsymbol{\tau}$	Torque
$c_{\sum}$	Mass-normalized total thrust	$\boldsymbol{F}$	Individual motor forces
$\boldsymbol{a}$	Individual motor speeds	$\boldsymbol{a}_{pwm}$	Individual motor 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_τ	Propeller constant: torque-to-thrust ratio
c	Size factor for randomization	λ	The sampled quadrotor
Learning Variables			
π	Base policy (our low-level controller)	ϕ	Adaptation module
μ	Intrinsics encoder	$\boldsymbol{e}_t$	Environmental parameters
$\boldsymbol{z}_t$	Intrinsics vector	$\boldsymbol{x}_t$	State vector
r_t	Reward		

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 it to focus on high-level mission tasks without concern for the intricacies of the underlying hardware. 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.

B. Learning an Adaptive Controller

An overview of our training strategy is illustrated in Figure 2. The controller learns to control randomly generated quadcopters to track diverse trajectories, which are generated with randomized motion primitives [24]. The training of the controller employs a dual strategy, combining reinforcement learning (RL) and behavior cloning (BC) from an expert model-based controller. We will provide details about each component during the training process in the following subsections.

1) Reinforcement Learning: As in our previous conference work [23], we use RMA [25] for the reinforcement learning part of our training. We do not make any changes to this part from our previous conference work, but review it here for completeness.

Our controller consists of a base policy π, an intrinsics encoder μ, and an adaptation module ϕ. At time t, the base policy π takes the current state $\boldsymbol{x}_t \in \mathbb{R}^{17}$ and the ground-truth

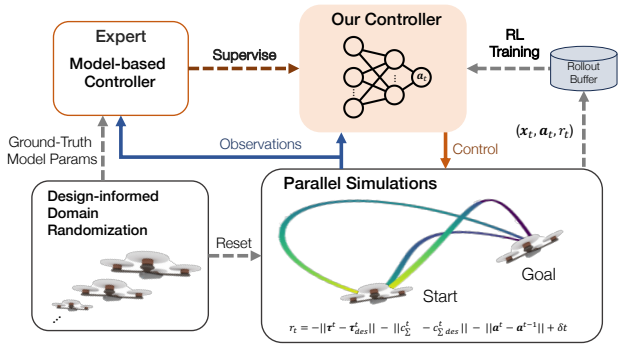


Fig. 2: An overview of the training process for our adaptive controller. The policy aims to track reference trajectories in simulation for various different quadcopters, whose parameters are determined through a domain randomization process that adheres to general quadcopter design principles. The training framework employs a hybrid approach, combining reinforcement learning (RL) with imitation learning derived from a model-based controller. The model-based controller, informed by ground-truth model parameters generated during the randomization process, offers expert supervision throughout the training phase.

intrinsics vector $\boldsymbol{z}_t \in \mathbb{R}^8$ to output the target motor speeds $\boldsymbol{a}_t \in \mathbb{R}^4$ for all individual motors. The intrinsics vector $\boldsymbol{z}_t$ is a low dimensional encoding of the environment parameters $\boldsymbol{e}_t \in \mathbb{R}^{34}$, which consist of model parameters or external disturbances that are key to adaptive control. We use the intrinsics encoder μ to compress $\boldsymbol{e}_t$ to $\boldsymbol{z}_t$. This gives us:

$$\boldsymbol{z}_t = \mu(\boldsymbol{e}_t) \tag{1}$$

$$\boldsymbol{a}_t = \pi(\boldsymbol{x}_t, \boldsymbol{z}_t) \tag{2}$$

The current state $\boldsymbol{x}_t$ includes the attitude matrix ($\mathbb{R}^9$), the mass-normalized thrust ($\mathbb{R}$), angular velocity ($\mathbb{R}^3$), commanded total thrust ($\mathbb{R}$) and commanded angular velocity ($\mathbb{R}^3$). The environmental parameter $\boldsymbol{e}_t$ includes mass, arm length, propeller constants (torque-to-thrust ratio and thrust-to-motorspeed-squared ratio), the diagonal entries of MMOI matrix ($\mathbb{R}^3$), body drag coefficients ($\mathbb{R}^3$), maximum motor rotation, motor effective factors ($\mathbb{R}^4$), mixer matrix ($\mathbb{R}^{4\times4}$), payload mass, and external torque ($\mathbb{R}^3$), which results in an 34 dimensional vector.

The low dimensionality (8) of $\boldsymbol{z}_t$ is determined empirically through a binary search on the dimension of $\boldsymbol{e}_t$, 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.

During deployment, we do not have access to $\boldsymbol{e}_t$ and hence we cannot directly measure $\boldsymbol{z}_t$ in the real world. Instead, we estimate it via the adaptation module ϕ, which uses the commanded actions and the measured sensor readings from the latest k steps to estimate it online during deployment as Equation 3. We can train this adaptation module in simulation using supervised learning because we have access to the ground truth intrinsics $\boldsymbol{z}_t$. We minimize the mean squared

error loss $\|\boldsymbol{z} - \hat{\boldsymbol{z}}\|^2$ when $\hat{\boldsymbol{z}}$ is estimated using sensor-action history of the vehicle tracking online generated random trajectories. The random trajectory tracking task can provide a set of rich excitation signals for the adaptation module to estimate $\hat{\boldsymbol{z}}$ using the sensor-action history.

The estimated $\hat{\boldsymbol{z}}_t$ along with the current state $\boldsymbol{x}_t$ is fed into our base policy π to output motor speed during deployment as Equation 4. More concretely,

$$\hat{\boldsymbol{z}}_t = \phi\big(\boldsymbol{x}_{t-k:t-1}, \boldsymbol{a}_{t-k:t-1}\big) \tag{3}$$

$$\boldsymbol{a}_t = \pi(\boldsymbol{x}_t, \hat{\boldsymbol{z}}_t) \tag{4}$$

2) Guiding search by imitating a model-based controller: We implement the base policy π and the intrinsics encoder μ as Multi-layer perceptrons and jointly train them end-to-end in simulation. The training is done by an integration of BC and model-free RL. In our approach, the expert controller is a model-based low-level controller. The key distinction from previous work combining reinforcement and imitation learning [26–28] 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 BC loss minimizes the mean squared error loss on actions:

$$L_{BC}(\pi) = \|\boldsymbol{a}_{exp} - \boldsymbol{a}\|^2 \tag{5}$$

Reinforcement learning maximizes the following expected return of the policy π:

$$R_{RL}(\pi) = \mathbb{E}_{\tau\sim p(\tau|\pi)}\left[\sum_{t=0}^{T-1}\gamma^t r_t\right], \tag{6}$$

where $\tau = \{(x_0, a_0, r_0), (x_1, a_1, r_1)...\}$ is the trajectory of the agent when executing the policy π, and $p(\tau|\pi)$ represents the likelihood of the trajectory under π.

Similarly to previous work, we adaptively change the relative weight between these two losses. The weight of the BC 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. The overall training framework seeks to maximize the overall reward of the policy π:

$$R(\pi) = (1-\alpha)R_{RL}(\pi) - \alpha L_{BC}(\pi) \tag{7}$$

$$\alpha = e^{-0.001 t_{\text{epoch}}} \tag{8}$$

This learning approach improves the training of the base policy compared to training solely with RL, as in our previous conference paper [23]. Figure 3 presents an ablation of the BC component of the loss. The policy trained with both BC and RL shows steady improvement in reward and episode length throughout the learning phase, maintaining an episode length close to 1 after approximately 10 million steps. In contrast, the RL-only baseline improves at a slightly slower rate, achieving reward similar to the BC-and-RL method at around 50 million steps. However, the RL-only baseline does not perform as consistently and begins to diverge after 50 million steps, losing much of its performance gains and eventually leading to crashes.

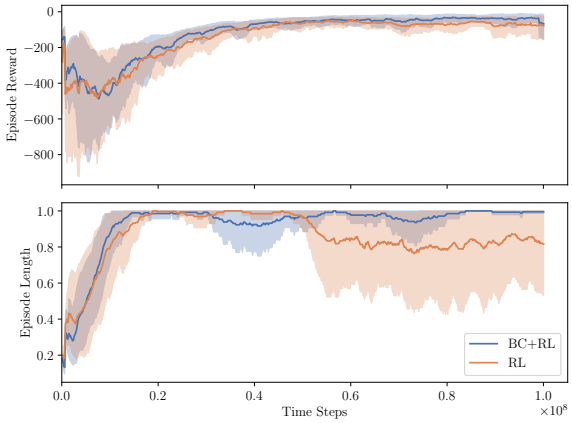


Fig. 3: Comparison of episode rewards (top) and episode lengths (bottom) for policies trained using both BC and RL, versus RL only, across 3 random seeds (with the bold line representing the mean and the shaded area representing the standard error). The episode length is normalized by dividing the vehicle's 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.

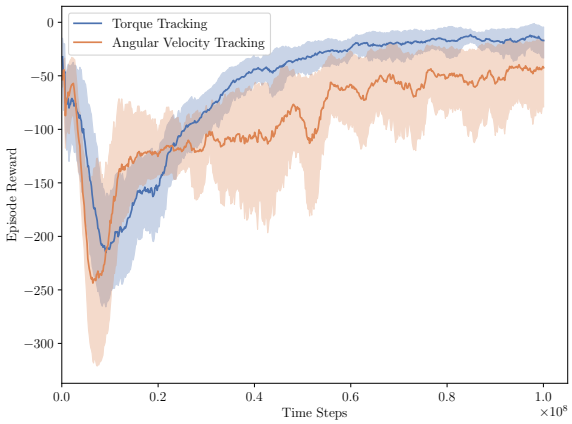


Fig. 4: We train the policy with torque tracking and with angular velocity tracking, and plot the episode reward except torque/angular velocity tracking penalty for 3 random seeds (bold line representing the mean and shaded area the standard error).

3) 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 Command Smoothing Penalty: $-\|\boldsymbol{a}^t - \boldsymbol{a}^{t-1}\|$
2) Survival Reward: δt
3) Mass-normalized Thrust Tracking Deviation Penalty: $-\|c^t_{\sum} - c^t_{\sum_{des}}\|$
4) Torque Tracking Deviation Penalty: $-\|\boldsymbol{\tau}^t - \boldsymbol{\tau}^t_{des}\|$

Where δt is the simulation step in the training episode. The mass-normalized thrust command $c_{\sum_{des}}$ is given by the high-level controller along with the commanded angular velocity $\boldsymbol{\omega}_{des}$. The commanded torque $\boldsymbol{\tau}_{des}$ is given by the rate control on the commanded angular velocity. In particular,

$$\boldsymbol{\tau}_{des} = JK(\boldsymbol{\omega}_{des} - \boldsymbol{\omega}) + \boldsymbol{\omega} \times (J\boldsymbol{\omega}) \tag{9}$$

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 smoothing penalty discourages high-frequency control commands, which can overheat the quadcopter's motors and damage the hardware system during deployment. 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.

While angular velocity is one of the high-level commands to track and one of our policy's observations during deployment, we prioritize rewarding torque tracking in our learning framework. Our experiments indicate that this design choice leads to noticeable improvements in learning speed, stability, and overall reward. Torque responds directly and immediately to applied motor speed commands, whereas angular velocity is derived through sequential integration of the commanded motor speed. Therefore, the torque tracking reward design can provide more direct reward after a corrective action, which is important for training a low-level high-bandwidth controller like ours, with such high frequency at 500Hz.

We compare the training curves of the policy trained with torque tracking and that with angular velocity tracking as shown in Figure 4, keeping all other reward design and hyperparameters the same. Specifically, we calculate the total reward per episode, excluding the torque / angular velocity tracking penalty, to evaluate the performance of the two policies. The torque tracking baseline improves throughout the training time steps, whereas the reward of the angular velocity tracking baseline grows with larger fluctuations and eventually ends at a lower reward.

4) Quadcopter Randomization: The training of our adaptive policy requires a wide spectrum of quadcopters, a challenge that we address through a carefully crafted randomization process. 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 quadcopters 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.

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: Thrust-to-Motorspeed-squared Ratio	[3.88e-8, 8.40e-6]	[3.24e-9, 1.02e-4]
Body drag coefficient	[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 c, which uniformly scales the size and motor strength of the quadcopter. We randomly sample c from the range of [0, 1]. The arm length is linearly scaled with c with minimum and maximum values from the training range of Table II.

$$l = c(l_{\max} - l_{\min}) + l_{\min} \tag{10}$$

Assuming a constant density and proportional scaling in all dimensions, the mass of the quadcopter is directly 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 airflow, scales with the square of the arm length.

We preserve the correlation by defining the mass, moment

of inertia and body drag coefficient as

$$\begin{cases} m & = c_m(m_{\max} - m_{\min}) + m_{\min} \\ c_m & = \frac{l^3 - l^3_{\min}}{l^3_{\max} - l^3_{\min}} \end{cases} \tag{11}$$

$$\begin{cases} J & = c_J(J_{\max} - J_{\min}) + J_{\min} \\ c_J & = \frac{l^5 - l^5_{\min}}{l^5_{\max} - l^5_{\min}} \end{cases} \tag{12}$$

$$\begin{cases} C_d & = c_{C_d}(C_{d_{\max}} - C_{d_{\min}}) + C_{d_{\min}} \\ c_{C_d} & = \frac{l^2 - l^2_{\min}}{l^2_{\max} - l^2_{\min}} \end{cases} \tag{13}$$

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.

$$C_F = C_{F_{\min}} \left(\frac{C_{F_{\max}}}{C_{F_{\min}}} \right)^c \tag{14}$$

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 flexibility and avoid adhering too strictly to the scaling rule, we introduce a uniformly distributed noise in the range of [-20%, 20%] to all parameters after they have been scaled with the size factor c.

Motor Efficiency Factor. We randomize the motor efficiency factor for each of the four rotors. For each rotor, the simulated motor speed is calculated by multiplying the intended speed by this factor. This is to simulate the motor ineffectiveness due to battery voltage drop, a partial motor failure, 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 sudden variations in the quadcopter parameters due to a sudden disturbance caused by an off-center payload or wind.

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 [29] to train and test our control policies. We implement the same high-level controller in [30] to generate high-level 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 are 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 5s of simulated time, with early termination if the quadcopter height drops below 2cm (equivalent to a crash to the ground), or the quadcopter's body rate exceeds 10rad/s. The control frequency of the policy is 500Hz, and the simulation step is 2ms. We additionally implement an observation 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]kg·m^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 [259e-6, 228e-6, 285e-6]kg·m^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 and our deployed policy, 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.

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 function 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. 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 z_t. For RL, we train the base policy and the environment encoder using PPO [31] for 100M steps in PyTorch. We use the reward described in Section II-B.3. Policy training takes approximately 1.5 hours on an ordinary desktop machine with 1 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) [32, 33] to convert trained models to MNN formats to optimize their inference speed and overhead.

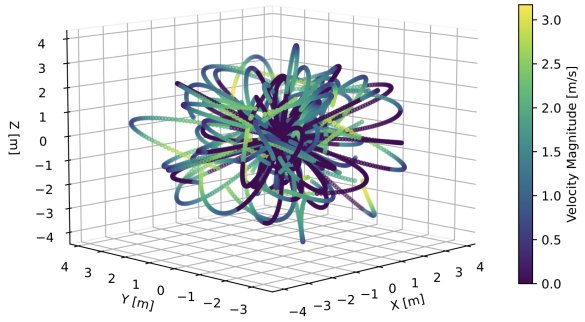


Fig. 5: Visualization of typical desired quadcopter trajectories in 3D space during simulated tests. The color gradient represents the magnitude of the velocity at each point along the trajectory. We sample 100 trajectories from the origin with the distribution defined by Table II. The initial conditions of all trajectories are hovering at origin.

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 II-B.4. The testing range is listed in Table II and a sample of typical desired trajectories is shown in Figure 5. We choose a nominal quadcopter model λ_{norm}, which is obtained by setting $c = 0.5$ when sampling without noise added.

We choose several different sets of high-level and low-level controllers as baselines from prior work. We choose *PID-PD*$_*$, with PID as the high-level controller and a low-level proportional-derivative controller with access to the ground truth model parameters of the sampled vehicle (*PD*$_*$); and *PID-PD*$_n$ with the low-level proportional-derivative controller only using parameters of λ_{norm} to control all the sampled quadcopters (*PD*$_n$). We choose these two baselines in particular because the *PID-PD*$_*$ can serve as the expected performance upper bound since it has access to ground-truth model parameters, and the *PID-PD*$_n$ can serve as the lower bound since it uses only the nominal model aiming to control all sampled vehicles. The two baselines can be used for the sanity check for our other proposed baselines and our approach, since their performance should be within the range bounded by these two methods.

In addition, we choose $\mathcal{L}_1$*-PD*$_n$ with $\mathcal{L}_1$ as a model-based adaptive high-level controller ([4–6]) and PD$_n$ as the low-level controller; and *PID-*$\mathcal{L}_1$ with $\mathcal{L}_1$ as the low-level adaptive controller. We design our method as a low-level controller under the assumption that the adaptation across the diverse parametric range as Table II is more effective at this control level. In other words, an adaptive high-level controller solely cannot sufficiently account for the model disparity in our problem, which aims to control quadcopters with significant difference in design and actuators. The purpose of $\mathcal{L}_1$ baselines is to validate our assumption by comparing their performance when the adaptation is at a different control level.

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 [24] with 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. This failure condition provides a clear distinction between a successful flight and a complete failure.

B. Results

The results of the simulation experiments are reported in Table III. We compare the five approaches under three metrics: (i) the success rate, (ii) the root-mean-square error (RMSE) in position tracking, and (iii) 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-PD*$_n$ with only access to the nominal model λ_{norm} 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

TABLE III: We choose 4 baselines: *PID-PD*$_*$, *PID-PD*$_n$, $\mathcal{L}_1$*-PD*$_n$ and *PID-*$\mathcal{L}_1$. 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 position and velocity RMSE between the actual quadcopter trajectory and the reference trajectory. The results are from 100 experiment for each baseline, and the position and velocity RMSE are obtained only from successful flights.

	Success Rate	Position RMSE $\pm\ \sigma$ (m)	Velocity RMSE $\pm\ \sigma$(m/s)
PID-PD$_n$	22%	0.510±0.372	0.845±1.066
$\mathcal{L}_1$-PD$_n$	62%	0.186±0.167	0.278±0.392
PID-$\mathcal{L}_1$	77%	0.221±0.242	0.357±0.481
PID-Ours	**100%**	**0.154±0.079**	**0.117±0.068**
PID-PD$_*$ (Expert)	100%	0.061±0.057	0.059±0.050

the ground truth parameters as *PID-PD*$_*$ but with adaptation to the unknown dynamics, the flight performance of $\mathcal{L}_1$ controllers significantly increases. However, with adaptation at high-level, the $\mathcal{L}_1$*-PD*$_n$ achieves a lower success rate than its counterpart *PID-*$\mathcal{L}_1$ with adaptation at low-level. Since tracking errors are only computed in successful runs, the $\mathcal{L}_1$*-PD*$_n$ 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.

PID-$\mathcal{L}_1$ is the strongest baseline after *PID-PD*$*$ in our simulations. However, its success depends on knowing the reference model λ_{norm}, while our method operates without this prior knowledge. Despite this, our method achieves a 100% success rate like *PID-PD*$*$, with only a slightly higher tracking error. This difference is expected, as *PID-PD*$*$ uses the ground-truth model parameters, whereas ours does not.

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 λ_{norm} now obtained at the mid-point of the training range. For ease of representation, we express λ and λ_{norm} in a numeric way, with its value equal to the scaling constant c. Therefore, $\lambda_{norm} = 0.5$ and $\lambda \in [0, 1]$ is the training set of Table II.

We use the metrics

$$\delta = \max |\lambda - \lambda_{norm}| \tag{15}$$

to define the extent of the range of sampled quadrotor parameters. In particular, when $\delta = 0$, the sampled quadrotor λ is the nominal model λ_{norm} with noises; when $\delta = 0.5$, $\lambda \in [0, 1]$ is the training set in Table II. We extend δ 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 6 provides a visualization

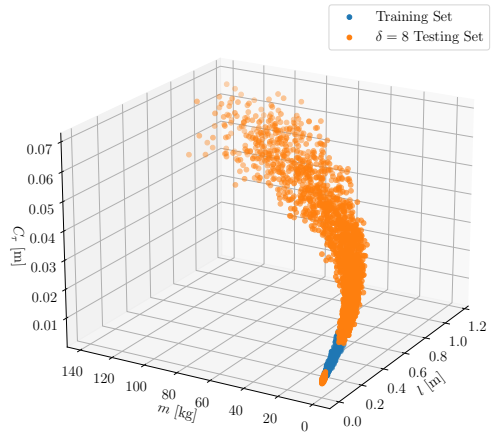


Fig. 6: Visualization of the differences between the training set and the $\delta = 8$ testing set. The scatter plot shows 2,000 randomly sampled quadcopters based on arm length l, mass m, and torque-to-thrust coefficient C_τ. The testing set exhibits a significantly wider distribution than the training set.

of the differences between the two sets. We randomly sample 2,000 quadcopters within the $\delta = 8$ range and within the training range, 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.

The success rate and the position and velocity tracking error distribution are reported in Figure 7. Our method achieve above 95% success rate until $\delta = 8$ where it drops to 84%. In contrast, all other baselines, except for *PID-PD*$_*$, exhibit significant performance degradation as the model mismatch from the nominal model increases. In addition, the average position tracking error at $\delta = 8$ of our method is still close to that at $\delta = 0$, with only 4.6% increase. Compared to the strongest baseline *PID-*$\mathcal{L}_1$ in Section IV-B whose position tracking error has grown by 481.1% compared to that at the nominal model.

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. This section aims to validate the effectiveness and robustness of our approach in real-world conditions.

A. Sim-to-Real Correlation

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 *PID-Ours* at 3 speed settings: *Slow*, *Medium* and *Fast*. Subsequently, we simulate the same flight paths of the same vehicle with our method

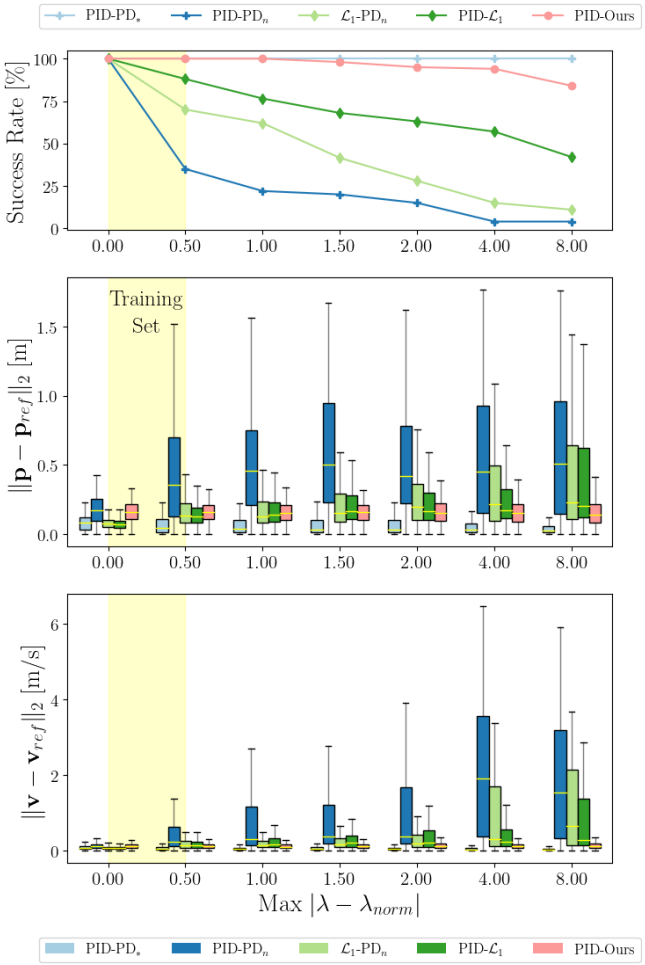


Fig. 7: We evaluate the performance of our method and all baselines on extended quadrotor parameters range unseen at training time. We use metrics $\delta = \max|\lambda - \lambda_{norm}|$, the maximum difference of all sampled quadcopters away from the nominal model, to define the quadrotor randomization range. We plot **Upper**: the success rate, **Middle**: the box plot of the position tracking error and **Lower**: 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 gray shaded area belong to the training range of Table II. Note that for better visualization, x-axis is not to scale.

again using the Flightmare simulator [29]. 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 8, 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 [34, 35], 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.

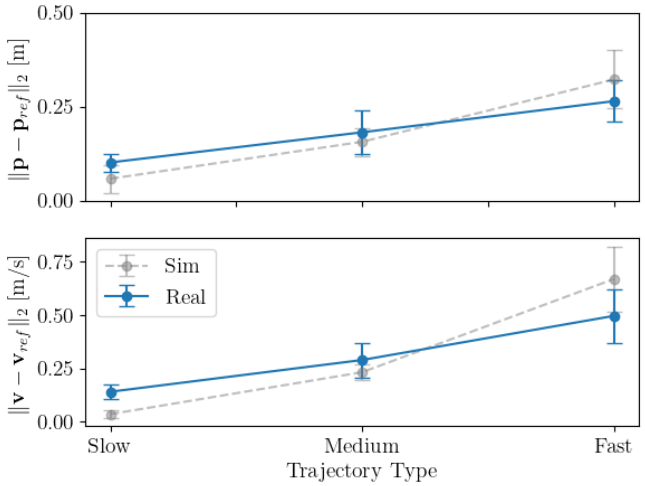


Fig. 8: To evaluate the sim-to-real gap, we control the large quadrotor with our proposed controller along circular trajectories at 3 speed settings: *Slow*, *Medium* and *Fast*. The figure illustrates the error bars for the distribution of position and velocity tracking errors. Both plots exhibits similar trend and magnitudes, suggesting that our simulator effectively reflects real-world dynamics and thus supporting the reliability of our results. These results are derived from 10 simulated flights and 3 real-world flights.

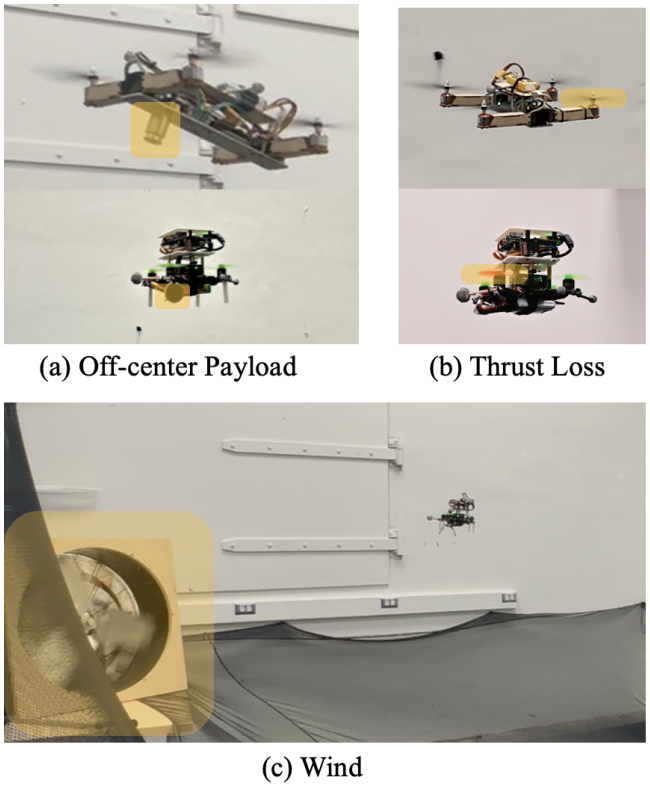
B. Baseline Comparison

We test our approach in the physical world and compare its performance to the *PID-PD*$_*$ controller that has access to the platform's parameters and has been specifically tuned to the platform with in-flight tests. In contrast, our approach has no knowledge whatsoever of the physical characteristics of the system and does not calibrate or fine-tune with real-world flight data. Our hardware experiments are designed to evaluate our method's capability to handle disturbances that are challenging to simulate. We test on the task of tracking a 1m circular trajectory with completion duration of 6s without any disturbances, under an off-center payload up to 20% of body mass which is attached to the farthest end on the body frame of the tested quadcopter, and under wind up to 3.5m/s. We also test on the task of taking off and hovering with one single motor experiencing 20% thrust loss. This is achieved by modifying the quadcopter's firmware to hard-code the hardware command sent to the affected motor so that the thrust produced by this particular actuator is always 80% of its desired value. We change the firmware to mimic the partial failure in the system in a controlled manner, instead of intentionally crashing the vehicle, to avoid actual damage and facilitate reproducibility. All experiment setup details are shown in Figure 9.

The high-level controller for both methods is a PID controller, the same as in Section III. The only variation in the two control frameworks is the low-level controller. Therefore, we compare the two approaches by their high-level command tracking performance. The comparison is conducted under two metrics: the average tracking error of the (i) mass normalized thrust and (ii) angular velocity given the high-level controller's commands. We define a failure as a situation where the human operator has to intervene to prevent the

TABLE IV: We compare the performance of our controller PID-Ours to the best baseline PID-PD$_*$ controller that uses accurate model information in Table III, in tasks of disturbance rejections that are hard to replicate in the simulation. Since the two control frameworks share the same high-level controller, we choose to focus on the high-level tracking performance as the comparison metrics. The comparison is run on the large quadcopter and the small quadcopter. We compare these approaches' performance tracking a circular trajectory at *Medium* speed under 3 tasks. **Disturb Free**: track the trajectory without any disturbances. **Off-center Payload**: track the trajectory under an unknown off-center payload. **Wind**: track the trajectory under wind. We also evaluate their performance on an additional task, **Thrust Loss**: take off and hover with one single motor experiencing 20% thrust loss. The results are from 3 experiments for each method per task.

	Vehicle	Low-level Controller	Thrust RMSE (m/s^2)	Average Angular Vel. RMSE (rad/s)
Disturb Free	small	PD$_*$	**0.132**	**0.339**
		Ours	0.280	0.721
	large	PD$_*$	**2.327**	**0.296**
		Ours	3.325	0.413
Wind	small	PD$_*$	2.477	1.365
		Ours	**1.659**	**0.429**
	large	PD$_*$	**2.916**	0.543
		Ours	3.549	**0.523**
Off-center Payload	small	PD$_*$	0.730	1.360
		Ours	**0.679**	**0.510**
	large	PD$_*$	3.420	0.697
		Ours	**2.808**	**0.456**
Thrust Loss	small	PD$_*$	0.689	1.408
		Ours	**0.403**	**0.576**
	large	PD$_*$	Fail	Fail
		Ours	**2.167**	**0.355**



(a) Off-center Payload (b) Thrust Loss

(c) Wind

Fig. 9: **(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.

quadcopter from crashing. The results of these experiments are reported in table IV. Note that in nearly all experiments, the thrust tracking RMSE for large quadrotor is much higher than that for the small quadrotor. This discrepancy arises because the large quadrotor has more powerful actuators, which can induce greater vibrations in the system, subsequently affecting the accelerometer readings. Therefore, this difference does not necessarily indicate that the thrust tracking for the large quadrotor is worse than for the small quadrotor. It is more appropriate to compare the performance of different methods within each platform, rather than across them.

Our approach and the *PD*$_*$ baseline perform similarly in disturbance-free experiments, with the *PD*$_*$ controller slightly outperforming ours. The latter difference in performance is justified since the *PD*$_*$ controller is specifically tuned for each quadcopter. Our method significantly outperforms the model-based *PD*$_*$ in both metrics in the presence of off-center payload and thrust loss. In particular, *PID-PD*$_*$ experiences a total failure in the case of thrust loss on the large quadrotor platform. Both disturbances create a large model mismatch from the nominal model that the *PID-PD*$_*$ uses. Our method is able to adapt to the mismatch well with a similar high-level command tracking error compared to that at the disturbance-free case. Conversely, the *PID-PD*$_*$ controller is not as adaptive. Indeed, its tracking error is up to 4.53 times higher for thrust tracking and up to 3.15 times for angular velocity tracking. Finally, the purpose of wind experiments is to evaluate our controller's performance under non-constant disturbances. Note that such time-varying disturbances were not present during training. Our controller is robust to wind disturbances with a comparable tracking performance as the *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.

VI. 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-to-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 partial motor failure. 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.

VII. 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 experimental testbed at the HiPeRLab is the result of contributions of many people, a full list of which can be found at `hiperlab.berkeley.edu/members/`.

References

[1] En-Hui Zheng, Jing-Jing Xiong, and Ji-Liang Luo. "Second order sliding mode control for a quadrotor UAV". In: *ISA Transactions* 53.4 (2014). Disturbance Estimation and Mitigation, pp. 1350–1356. ISSN: 0019-0578. DOI: https://doi.org/10.1016/j.isatra.2014.03.010. URL: https://www.sciencedirect.com/science/article/pii/S0019057814000512.

[2] Daniel Mellinger, Nathan Michael, and Vijay Kumar. "Trajectory generation and control for precise aggressive maneuvers with quadrotors". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 664–674.

[3] Iven M.Y. Mareels et al. "Revisiting the Mit Rule for Adaptive Control". In: *IFAC Proceedings Volumes* 20.2 (1987), pp. 161–166. ISSN: 1474-6670.

[4] Chengyu Cao and Naira Hovakimyan. "Design and analysis of a novel L1 adaptive control architecture with guaranteed transient performance". In: *IEEE Transactions on Automatic Control* 53.2 (2008), pp. 586–591.

[5] Naira Hovakimyan and Chengyu Cao. *L1 adaptive control theory: Guaranteed robustness with fast adaptation*. SIAM, 2010.

[6] Drew Hanover et al. "Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors". In: *IEEE Robotics and Automation Letters* 7.2 (2021), pp. 690–697.

[7] Jintasit Pravitra et al. "L1-Adaptive MPPI Architecture for Robust and Agile Control of Multirotors". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 7661–7666.

[8] Karime Pereida and Angela P Schoellig. "Adaptive model predictive control for high-accuracy trajectory tracking in changing conditions". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 7831–7837.

[9] Jemin Hwangbo et al. "Control of a quadrotor with reinforcement learning". In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103.

[10] William Koch et al. "Reinforcement learning for UAV attitude control". In: *ACM Transactions on Cyber-Physical Systems* 3.2 (2019), pp. 1–21.

[11] Yunlong Song et al. "Autonomous drone racing with deep reinforcement learning". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 1205–1212.

[12] Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. "A benchmark comparison of learned control policies for agile quadrotor flight". In: *arXiv preprint arXiv:2202.10796* (2022).

[13] Elia Kaufmann et al. "Champion-level drone racing using deep reinforcement learning". In: *Nature* 620.7976 (2023), pp. 982–987. DOI: 10.1038/s41586-023-06419-4. URL: https://doi.org/10.1038/s41586-023-06419-4.

[14] Andrea Tagliabue et al. *Demonstration-Efficient Guided Policy Search via Imitation of Robust Tube MPC*. 2021. arXiv: 2109.09910 [cs.RO].

[15] Tong Zhao, Andrea Tagliabue, and Jonathan P. How. *Efficient Deep Learning of Robust, Adaptive Policies using Tube MPC-Guided Data Augmentation*. 2023. arXiv: 2303.15688 [cs.RO].

[16] Michael O'Connell et al. "Neural-Fly enables rapid learning for agile flight in strong winds". In: *Science Robotics* 7.66 (2022), eabm6597.

[17] Kevin Huang et al. “DATT: Deep Adaptive Trajectory Tracking for Quadrotor Control”. In: *arXiv e-prints*, arXiv:2310.09053 (Oct. 2023), arXiv:2310.09053. DOI: 10.48550/arXiv.2310.09053. arXiv: 2310.09053 [cs.RO].

[18] Gilbert Feng et al. *GenLoco: Generalized Locomotion Controllers for Quadrupedal Robots*. 2022. arXiv: 2209.05309.

[19] Elia Kaufmann et al. “Deep Drone Acrobatics”. In: *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA, 2020. DOI: 10.15607/RSS.2020.XVI.040.

[20] Jonas Eschmann, Dario Albani, and Giuseppe Loianno. “Learning to Fly in Seconds”. In: *IEEE Robotics and Automation Letters* (2024), pp. 1–8. DOI: 10.1109/LRA.2024.3396025.

[21] Artem Molchanov et al. “Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 59–66.

[22] Philip Becker-Ehmck et al. “Learning to fly via deep model-based reinforcement learning”. In: *arXiv preprint arXiv:2003.08876* (2020).

[23] Dingqi Zhang et al. “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. DOI: 10.1109/ICRA48891.2023.10160836.

[24] Mark W. Mueller, Markus Hehn, and Raffaello D’Andrea. “A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation”. In: *IEEE Transactions on Robotics* 31.6 (2015), pp. 1294–1310. DOI: 10.1109/TRO.2015.2479878.

[25] Ashish Kumar et al. “Rma: Rapid motor adaptation for legged robots”. In: *RSS: Robotics Science and Systems* (2021).

[26] Zhaoming Xie et al. “Learning locomotion skills for cassie: Iterative design and sim-to-real”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 317–329.

[27] Huang Huang et al. “Manipulator as a Tail: Promoting Dynamic Stability for Legged Locomotion”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 9712–9719.

[28] Antonio Loquercio, Ashish Kumar, and Jitendra Malik. “Learning visual locomotion with cross-modal supervision”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 7295–7302.

[29] Yunlong Song et al. “Flightmare: A Flexible Quadrotor Simulator”. In: *Proceedings of the 2020 Conference on Robot Learning*. 2021, pp. 1147–1157.

[30] Mark Wilfried Mueller. “Multicopter attitude control for recovery from large disturbances”. In: *arXiv preprint arXiv:1802.09143* (2018).

[31] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[32] Xiaotang Jiang et al. *MNN: A Universal and Efficient Inference Engine*. 2020. arXiv: 2002.12418 [cs.CV].

[33] Chengfei Lv et al. “Walle: An End-to-End, General-Purpose, and Large-Scale Production System for Device-Cloud Collaborative Machine Learning”. In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, July 2022, pp. 249–265. ISBN: 978-1-939133-28-1. URL: https://www.usenix.org/conference/osdi22/presentation/lv.

[34] David Freedman, Robert Pisani, and Roger Purves. “Statistics (international student edition)”. In: *Pisani, R. Purves, 4th edn. WW Norton & Company, New York* (2007).

[35] Jacob Benesty et al. “Pearson correlation coefficient”. In: *Noise reduction in speech processing*. Springer, 2009, pp. 37–40.