Learning on the Fly: Rapid Policy Adaptation via Differentiable Simulation

Anonymous Author(s)
Affiliation
Address
email

Abstract: Learning control policies in simulation enables rapid, safe, and cost-effective development of advanced robotic capabilities. However, transferring these policies to the real world remains difficult due to the sim-to-real gap, where unmodeled dynamics and environmental disturbances degrade performance. Current approaches, such as domain randomization, improve robustness but cannot capture the full range of real-world variations. In this work, we approach these problems from a different perspective. Instead of relying on diverse training conditions, we focus on adapting the learned policy directly in the real world and in real time. To achieve this, we propose a policy adaptation framework that combines online residual dynamics learning with real-time policy fine-tuning. Starting from a lightweight rigid-body model, our framework refines the dynamics using real-world data, thereby augmenting the simulation with previously unmodeled effects and disturbances such as unknown payloads and wind. The updated simulation is then used within a differentiable simulation framework to compute policy gradients efficiently, enabling real-time updates with minimal sample and computational cost. All components of our system are designed for rapid adaptation, enabling the policy to adjust to unseen disturbances within *5 seconds*. Our approach supports both state-based and vision-based inputs. It is validated in simulation as well as on real quadrotors under diverse disturbance conditions, consistently outperforming both model-based controllers and learning-based methods trained with domain randomization.

1 Introduction

Robot learning through simulation has seen great success in recent years, thanks to the rapid improvements in computer hardware and advancements in efficient physics simulation [1]. Simulation provides a fast, safe, and cost-effective way to collect data and train policies, enabling experiments that would be impractical or unsafe in the real world. However, transferring control policies learned purely in simulation to physical systems is challenging. Even with high model fidelity in simulation, many system parameters are difficult to identify, making accurate alignment between simulation and reality hard to achieve. In addition, unmodeled effects such as aerodynamic turbulence, sensor noise, and actuator delays further complicate the real-world dynamics. The resulting mismatch, known as the sim-to-real gap [2], remains a central obstacle to deploying learning-based controllers in the real world. Bridging this gap is essential to retain the advantages of simulation while ensuring that policies perform reliably under the complexity and variability of real-world conditions. Domain randomization is a common strategy to address this issue [3, 4, 5, 6], in which simulation parameters such as dynamics, sensor noise, and visual appearance are varied during training to expose the policy to a wide range of possible deployment scenarios. By learning in diverse conditions, the agent develops robust policies that are less likely to overfit to the specific characteristics of a single environment. However, domain randomization cannot anticipate every possible real-world condition [7]. When the environment shifts beyond the randomized distribution, policy performance might strongly degrade. Broadening the range of randomization increases coverage but also inflates

Submitted to the 9th Conference on Robot Learning (CoRL 2025) Workshop. Do not distribute.

the exploration space, slowing convergence and reducing learning efficiency [8]. Beyond domain randomization, Real2Sim2Real pipelines [9, 10] have shown strong sim-to-real transfer by refining simulation models with real-world data before retraining policies for deployment. While offline residual learning is effective in improving transfer, these methods require extensive offline data collection and long retraining cycles, making them unsuitable for online, real-time adaptation.

In this work, we try to approach these problems from another perspective: we propose to *rapidly adapt the policy* to unknown external disturbances in the real world and in real time. We evaluate our method on an agile quadrotor platform, whose nonlinear dynamics and sensitivity to aerodynamic effects make it a challenging benchmark for adaptive control [11]. The core insight of the proposed framework is to unify online residual dynamics learning with real-time policy adaptation inside a differentiable simulation framework. All system components are designed to update model estimation and policy *as quickly as possible*, ideally within a few seconds, during runtime. In this way, the policy can "overfit" rapidly to the current scenario, which paradoxically enables it to become more "generalizable" across diverse real-world conditions.

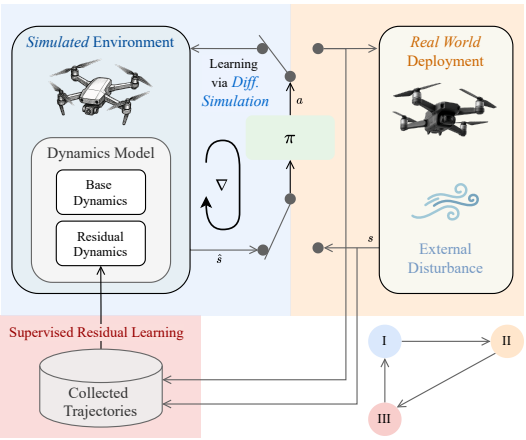


Figure 1: Overview of three alternating stages (I, II, III) in our proposed approach. (I) Based on the latest simulation dynamics, a policy is rapidly trained via differentiable simulation. (II) The policy is deployed in the real world, and the trajectories are stored. (III) Residual dynamics are trained using the stored real-world data to refine the simulation dynamics.

For our pipeline, we start with a lightweight rigid-body dynamics model and continuously refine it by learning residual dynamics from real-world flight data. By embedding the residual-augmented model in a differentiable simulator, we improve the fidelity of the simulated dynamics through online refinement, enabling more accurate and sample-efficient adaptation. Another key innovation is our alternating optimization scheme, where policy learning and residual model learning are interleaved so that each batch of real-world data is used efficiently for both dynamics refinement and control improvement. All of these components ensure the simulation is aligned with reality and enables real-time adaptation to unknown disturbances, even controlling directly from perceptual input. We evaluate the proposed framework in both simulation and real-world experiments on an agile quadrotor platform, considering a range of environmental disturbance conditions. In state-based control tasks such as hovering, our method attains an average error of 0.105 m, an 81% reduction over $\mathcal{L}_1$-MPC (0.552 m) and 55% over DATT [12] (0.231 m), while ensuring stable flight under modeling errors and out-of-distribution disturbances. In vision-based tasks, our framework achieves similar gains, demonstrating that real-time adaptation remains effective under partial or noisy observations, something unattainable with classical control methods in the absence of state estimation.

Contributions. We propose an online adaptive control framework that combines residual dynamics learning with real-time policy adaptation in a differentiable simulation environment, enabling real-world policy adaptation **within 5 seconds**. Starting from a simplified dynamics model, the framework refines the dynamics online with real-world data to capture mismatches and unpredictable disturbances. We propose an alternating optimization scheme that interleaves policy learning and dynamics learning, ensuring that each real-world sample improves both components. The framework supports both state-based and vision-based inputs, and we validate its effectiveness through extensive simulation and real-world quadrotor experiments, where it outperforms both classical and learning-based controllers under large unseen disturbances. Together, our framework shows that policies can learn and adapt within seconds in the real world, reducing the reliance on domain randomization, which can fail to capture real-world out-of-distribution complexity.

2 Related Work

Policy Learning in Differentiable Simulation. Differentiable simulation uses smooth, differentiable dynamics and rewards to enable policy learning via first-order gradients [13], offering substantial gains in sample efficiency and training time over traditional RL [14]. It has been applied to direct policy parameterizations, such as parametric curve frequencies for swimming robots [15] and sinusoidal policies for robotic cutting [16], as well as to neural-network policies. However, unstable gradients often limit applications to short-horizon tasks with simplified contacts and restricted start-state variation [17, 18, 19, 20]. To address this, prior work has explored enhancements such as early-stopping simulations at contact events, truncated BPTT [21], and reward augmentation with a learned critic [22, 23].

Aligning Simulation with the Real-World. Closing the sim-to-real gap requires quantifying the misalignment between simulation and real-world dynamics, typically through system identification or residual dynamics learning. System identification estimates parameters of an analytical dynamics model from input–output data [24, 25], but its representation capacity is limited to the modeled system [26], making it less effective for capturing complex dynamics and disturbances [27]. Residual dynamics learning addresses this by directly modeling the discrepancy between analytical predictions and real-world measurements. It has been applied to improve quadrotor odometry and tracking [28, 29, 30], learn motor delays in quadrupeds [10], and predict residual forces in soft robots [31]. In [32], it has been demonstrated that the perception encoding can also be aligned between simulation and the real world using a contrastive learning method.

Learning-Based Adaptive Control. Residual dynamics models have been used for online disturbance estimation, via offline-trained networks [33], Gaussian Processes [30], or differentiable simulation–based system identification [34]. These methods mainly augment optimization-based controllers like MPC, which rely on full state information and do not directly extend to vision-based control. Neural policies have also been conditioned on disturbance estimates [12, 35, 36], but since they are trained offline in randomized simulations and remain fixed at deployment, they struggle with domain shifts and unseen conditions [37, 12].

3 Methodology

Our approach consists of two phases: policy pretraining and online adaptation. As shown in Fig. 2, during pretraining, we train a base policy using the low-fidelity dynamics model without residual dynamics or domain randomization, which is used as the initial policy for online adaptation. For online adaptation, two independent learning processes run in parallel. The residual dynamics learning loop continuously refines a residual dynamics network using a rolling history buffer of recorded quadrotor states and actions. The policy adaptation loop embeds the latest residual dynamics network parameters into the differentiable simulation pipeline and performs fast policy adaptation.

Policy Adaptation via Differentiable Simulation. We model the quadrotor as a discrete-time dynamical system with continuous state and action spaces $\mathcal{X}$ and $\mathcal{U}$, respectively. The system evolves according to the hybrid dynamics model $f_{\text{hybrid}} : \mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$ which comprises the analytical and learned residual components, and describes the system evolution $x_{t+1} = f_{\text{hybrid}}(x_t, u_t)$ over time. At time step t, an observation model $h : \mathcal{X} \mapsto \mathcal{O}$ generates an observation $o_t = h(x_t) \in \mathcal{O}$ from the state x_t, and is passed as input to a deterministic and differentiable policy network $\pi_\phi : \mathcal{X} \mapsto \mathcal{U}$ which outputs an action $u_t = \pi_\phi(o_t)$, and finally a deterministic, smooth and differentiable reward function $r : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$ emits a reward $r_t = r(x_t, u_t)$ based on the state-action pair.

Policy Optimization Using Analytical Gradients. The policy learning objective is to maximize the cumulative task reward $\mathcal{R}(\phi)$ over an N-step rollout under the policy parameters ϕ via

$$\max_{\phi} \mathcal{R}(\phi) = \sum_{t=0}^{N-1} r(x_t, u_t) = \sum_{t=0}^{N-1} r(x_t, \pi_\phi(h(x_t))). \tag{1}$$

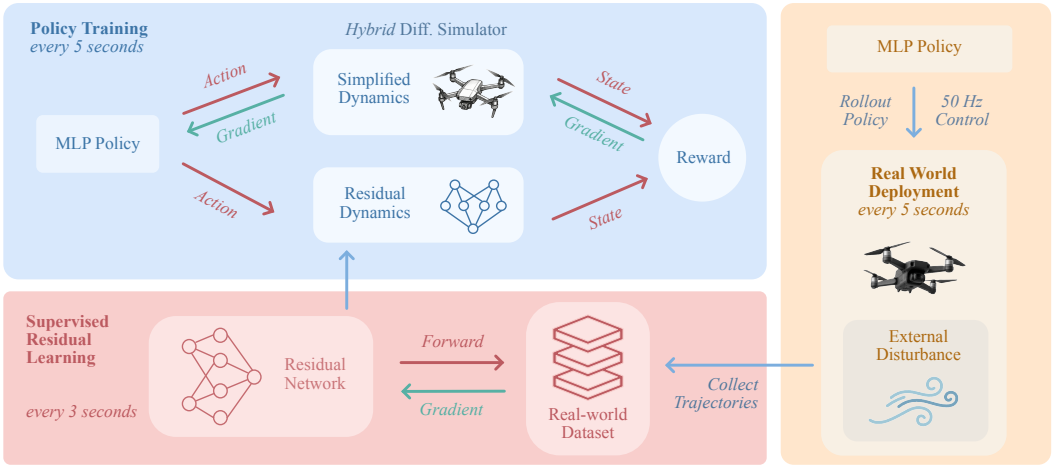


Figure 2: Detailed illustration of the information flow both within and between the three interleaved components of our proposed approach.

By leveraging the differentiable dynamics and reward structure, we can obtain first-order analytical policy gradients of the objective (1) via Back-Propagation Through Time (BPTT) (see [38] for a full derivation). The policy gradient and the update rule for the policy parameters ϕ are given by

$$\nabla_\phi R(\phi) = \frac{1}{N}\sum_{t=0}^{N-1}\left(\sum_{i=1}^{t}\frac{\partial r_t}{\partial x_t}\prod_{j=1}^{t}\left(\frac{dx_j}{dx_{j-1}}\right)\frac{\partial x_i}{\partial \phi} + \frac{\partial r_t}{\partial u_t}\frac{\partial u_t}{\partial \phi}\right), \qquad \phi_{k+1} = \phi_k + \alpha \nabla_\phi R(\phi_k), \tag{2}$$

where $\frac{dx_j}{dx_{j-1}}$ is the derivative matrix of the system dynamics f_{hybrid}, and α is the learning rate. We build upon an existing open-source differentiable simulator for quadrotors [14] written entirely in JAX to leverage both its automatic-differentiation framework for computing the analytical policy gradients and GPU-accelerated computing for efficient parallel simulation.

Residual Dynamics Learning. Given the quadrotor state $\boldsymbol{x}^\top = [\boldsymbol{p}^\top, \boldsymbol{R}^\top, \boldsymbol{v}^\top]$ consisting of position $\boldsymbol{p} \in \mathbb{R}^3$, rotation matrix $\boldsymbol{R} \in \mathrm{SO}(3)$, linear velocity $\boldsymbol{v} \in \mathbb{R}^3$, along with the normalized collective thrust command $c \in \mathbb{R}$ and the body rates command $\boldsymbol{\omega}_{\text{cmd}} \in \mathbb{R}^3$, we form the concatenated input vector $[\boldsymbol{x}^\top, \boldsymbol{u}^\top] = [\boldsymbol{p}^\top, \boldsymbol{R}^\top, \boldsymbol{v}^\top, c, \boldsymbol{\omega}_{\text{cmd}}^\top] \in \mathbb{R}^{19}$. An MLP network f_{res} with two 128-dimensional hidden layers is trained to predict the residual acceleration $\boldsymbol{a}_{\text{res}} \in \mathbb{R}^3$, defined as the difference between the ground-truth acceleration $\boldsymbol{a}_{\text{gt}} \in \mathbb{R}^3$ measured on the real system and the theoretical acceleration $\hat{\boldsymbol{a}} = f_{\text{a}}([\boldsymbol{x}, \boldsymbol{u}]) \in \mathbb{R}^3$ from the analytical dynamics model f_{a} (3). The residual acceleration training targets are computed as $\boldsymbol{a}_{\text{res}} = \boldsymbol{a}_{\text{gt}} - \hat{\boldsymbol{a}}$. Given a batch of $|\mathcal{B}|$ samples $\{[\boldsymbol{x}, \boldsymbol{u}]^i, \boldsymbol{a}_{\text{res}}^i\}_{i \in \mathcal{B}}$, we train the model by minimizing the loss function $\mathcal{L}_{\text{res}}$ via $\min_\theta \mathcal{L}_{\text{res}} = \min_\theta \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|} ||\boldsymbol{a}_{\text{res}}^i - f_{\text{res}}([\boldsymbol{x}, \boldsymbol{u}]^i; \boldsymbol{\theta})||^2 + \beta \sum_{l=1}^{L} ||W^l||_2^2$. The loss comprises a standard MSE term and a spectral norm regularization term, where the latter has been shown to improve generalization beyond the training distribution [39] by regulating the network's Lipschitz constant [33]. Here, W^l is the weight matrix of the l-th network layer, and β controls the regularization strength. The loss is minimized using the Adam optimizer [40].

Design Choices for Maximum Runtime Efficiency. During forward simulation, we use a hybrid dynamics model f_{hybrid} obtained by additively combining the analytical f_{a} and learned residual f_{res} dynamic models. Here, we use a simple, low-fidelity analytical dynamics model given by (3) which models the quadrotor as a point-mass. The resulting acceleration given a state and action input pair is computed as $\hat{\boldsymbol{a}}_{\text{hybrid}} = \hat{\boldsymbol{a}} + \hat{\boldsymbol{a}}_{\text{res}}$, where $\hat{\boldsymbol{a}}_{\text{res}}$ is the network prediction. Finally, the quadrotor states are simulated via Runge-Kutta 4 time-integration at 50 Hz of the dynamics (3) using $\hat{\boldsymbol{a}}_{\text{hybrid}}$.

While the hybrid dynamics model f_{hybrid} composed of differentiable analytical and learned components remains overall fully differentiable, we only perform gradient backpropagation through the analytical dynamics model and not the frozen network to obtain the policy gradients. This was in-

spired by prior work in policy learning using differentiable simulation for both quadruped [41] and quadrotor [14] control, which showed that combining accurate forward dynamics simulation with the backpropagation of a surrogate gradient based on a simpler dynamics model achieves faster runtime without impacting the resulting policy performance. We analyze and justify the above design choices through simulated experiments, and present and discuss the results in Appendix C.

Full vs. Residual Policy Adaptation. We compare two existing methods of adapting a pretrained policy: full vs. residual adaptation [42]. Full adaptation involves updating all parameters of the policy network, similar to [43], whereas residual adaptation freezes all pretrained parameters and instead adapts an additive residual policy module, which forms a lower-dimensional trainable parameter space. The latter has been shown to achieve more memory and parameter-efficient policy adaptation for applications such as autonomous racing and quadruped control [44, 23]. We implement the residual policy module as low-rank weight matrices added in parallel to the pretrained policy network, similar to the low-rank adaptation (LoRA) implementation commonly used to fine-tune Large-Language Models [45]. We seek to understand whether residual policy adaptation enables better sample efficiency and learning stability than full adaptation in low-data regimes during online policy adaptation, where relatively few policy rollout samples are generated and used.

4 Experiments and Results

4.1 Experimental Setup

Task Definitions. We evaluate our approach on stabilizing hover and trajectory tracking tasks for the quadrotor platform. For stabilizing hover, the policy is required to regulate the quadrotor state towards a goal position $\boldsymbol{p}_{\text{des}}$ and maintain it at all times, which is non-trivial given the quadrotor's non-linear and unstable dynamics (3). We evaluate both a state-based control policy which at each time step receives the observation $\boldsymbol{o} = [\boldsymbol{p}, \boldsymbol{R}, \boldsymbol{v}]^\top$, and an end-to-end vision-based policy which only receives the projected pixel coordinates of 7 visual features from the past 5 time steps and the last 3 actions. Our training setup closely follows the open-source environment setup in [14]. Trajectory tracking instead requires following a reference trajectory defined as a time-parameterized sequence of quadrotor states, which is generated to be smooth up to the acceleration level. We use two such trajectories, *Circle* and *Figure-8*, for this task in a state-based training setting. The reward definitions for both tasks and more details on the reference trajectories are included in Appendix B.

Pretraining Phase. We parameterize the policy as an MLP with two 512-dim hidden layers. For both state-based hovering and trajectory tracking, we train the base policy from random initialization for 300 epochs across 100 parallel environments. Each epoch lasts 3 seconds or 150 simulation steps. We then train the partially initialized policy for 500 epochs across 300 parallel environments.

Online Adaptation Phase. The quadrotor states and actions are continuously recorded into a rolling history buffer at 50 Hz and are used to train the residual dynamics network. For the stabilizing hover and trajectory tracking tasks, we use history buffer sizes of 100 and 250, which are equivalent to 2 and 5 seconds of past quadrotor states and actions, respectively. For residual dynamics learning, we continuously refine an ensemble of 3 networks initialized with different random seeds, and use the empirical mean prediction from all models as the final predicted residual acceleration for a given input. Empirically, we found this to effectively reduce the prediction variance arising from epistemic uncertainty due to the limited samples in the data buffer. We run residual dynamics learning every 3 seconds and train the networks for 100 epochs, which takes approximately **2 seconds**. We run policy adaptation every 5 seconds and train the policy with 10 parallel simulated environments for 30 epochs, which takes approximately **1.5 seconds**. All experiments are run on a workstation equipped with an Nvidia RTX 4090 GPU with 24 GB of VRAM.

Baselines Methods. We compare against a state-of-the-art learning-based adaptive control method, Deep Adaptive Tracking Control (DATT) [12], which uses model-free RL with domain randomization and online $\mathcal{L}_1$ adaptive control-based disturbance estimation. For quadrotor control, this method has been shown to outperform Rapid Motor Adaptation (RMA) [36], which is a similar approach but

Table 1: Average steady-state error (in m) from the hovering target across 8 rollouts. The errors of the two best-performing methods for each disturbance condition are highlighted.

Method	No disturbance	Small disturbance	Large disturbance
$\mathcal{L}_1$-MPC	0.091 ± 0.052	0.134 ± 0.073	0.552 ± 0.130
DATT	0.013 ± 0.004	0.009 ± 0.005	0.231 ± 0.004
Ours	0.015 ± 0.001	0.008 ± 0.002	0.105 ± 0.007
Ours (Residual)	0.023 ± 0.002	0.015 ± 0.004	0.125 ± 0.002

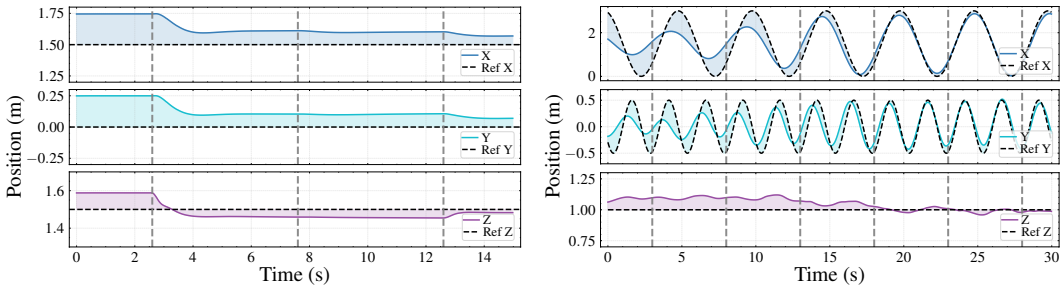
instead uses a learned encoder for disturbance estimation. We used the open-source implementation of [46] and the exact same training procedure and hyperparameter settings to train both state-based hovering and trajectory tracking policies using PPO for 20 million simulation steps. Using their original domain randomization implementation, we simulated 3-dimensional acceleration disturbances as random walks within the bounds +/- $[1, 1, 1]\,\mathrm{m/s^2}$. We also compare against an adaptive Nonlinear MPC controller ($\mathcal{L}_1$-MPC) as implemented in [46], which uses a Model Predictive Path Integral (MPPI) formulation and the same $\mathcal{L}_1$ adaptive control-based disturbance estimation as in DATT.

4.2 Experimental Results

For evaluation, we used a realistic quadrotor simulator [11] equipped with the BEM model for aerodynamic effects and high-frequency simulations of the controller dynamics. We simulated three levels of constant, uniform acceleration disturbances: $[0, 0, 0]\,\mathrm{m/s^2}$ (*none*), $[0.5, 0.5, 0.5]\,\mathrm{m/s^2}$ (*small*), and $[2, 2, 2]\,\mathrm{m/s^2}$ (*large*). The first two conditions are within the domain randomization range used for DATT training, whereas the third condition was deliberately chosen to be out of distribution to evaluate its generalization capabilities. The policy outputs actions consisting of mass-normalized collective thrust and desired body rates to the on-board flight controller at $50\,\mathrm{Hz}$.

Performance Comparison to Baseline Approaches. For the state-based stabilizing hover task, we ran each method under all disturbance conditions from a set of 8 different starting positions around the hovering target, and used the final steady-state error as the performance metric. The baseline methods were run for 10 seconds to allow transient adaptation effects to settle, while our method (both state and vision-based) was run for 30 seconds to allow a few learning steps to take place. As summarized in Tab. 1, results show that our method consistently exhibits superior or comparable performance to the baselines. Fig. 3a illustrates that our method rapidly adapts the policy to compensate for the *large* disturbances within 2-3 adaptation steps (10-15 seconds of wall time). DATT performs well under both the *none* and *small* disturbance scenarios which are both within its training distribution, but struggles to adapt to the larger, out-of-distribution disturbance.

We exclude the baseline methods from the vision-based hovering comparison, as it would require significant modifications to their algorithms. As shown in Tab. 3, our vision-based approach resulted in larger errors than our state-based approach. We empirically observed that adaptation of the vision-based policy is less stable than the state-based counterpart and may require more policy



(a) State-based hovering under *large* disturbance.

(b) *Figure-8* tracking under *small* disturbance.

Figure 3: Real-time policy adaptation using our proposed approach in simulation. Vertical dashed lines indicate policy update steps. Shaded regions represent the error from the reference.

Table 2: Average tracking errors (in m) for two different trajectories (*Circle* and *Figure-8*). The errors of the two best-performing methods for each disturbance condition are highlighted.

Trajectory	Method	No disturbance	Small disturbance	Large disturbance
Circle	$\mathcal{L}_1$-MPC	0.113 ± 0.027	0.096 ± 0.063	0.410 ± 0.155
	DATT	0.058 ± 0.016	0.040 ± 0.024	*crash*
	Ours	0.167 ± 0.048	0.135 ± 0.101	0.349 ± 0.175
	Ours (Residual)	0.129 ± 0.051	0.159 ± 0.043	0.326 ± 0.061
Figure-8	$\mathcal{L}_1$-MPC	0.109 ± 0.063	0.121 ± 0.025	0.281 ± 0.097
	DATT	0.078 ± 0.037	0.082 ± 0.046	*crash*
	Ours	0.068 ± 0.040	0.045 ± 0.03	0.137 ± 0.098
	Ours (Residual)	0.069 ± 0.047	0.059 ± 0.043	0.110 ± 0.037

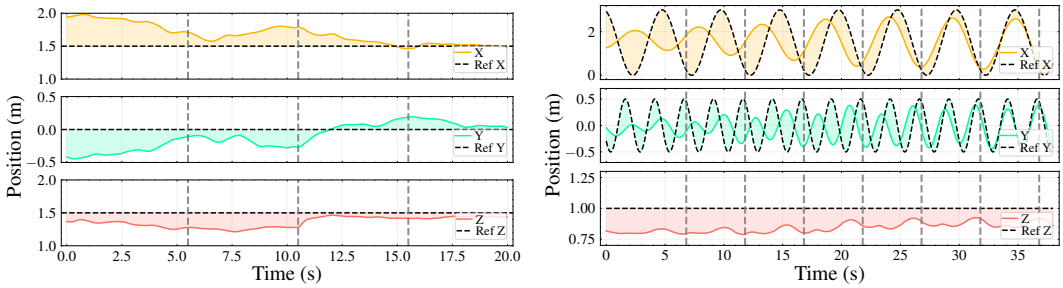
learning epochs or update steps, most likely due to the lack of explicit state information and sample inefficiency in learning vision-based control.

For trajectory tracking, we recorded 60-second rollouts and computed the average tracking error (m) within the last 10-second window as the performance metric. As shown in Tab. 2, our approach achieves comparable performance to the baselines for both *Circle* and *Figure-8* across all disturbance conditions. Fig. 3b illustrates that the large initial sim-to-real gap due to policy pretraining using the low-fidelity dynamics model (3) already significantly impacts tracking accuracy even under the *small* disturbance condition. However, our method is able to rapidly adapt and achieve much improved tracking accuracy after only 4 policy update steps (20 seconds of wall time). For DATT, consistent with findings from state-based hovering, it fails to generalize to the out-of-distribution disturbances and results in crashes for both reference trajectories. Finally, we observe that using residual policy adaptation with our approach achieved comparable performances to using full adaptation across all tasks and disturbance conditions. Therefore, while residual adaptation may not provide significant benefit regarding sample efficiency for policy learning, it still remains as a parameter-efficient, and hence, memory-efficient paradigm to adapt or fine-tune a pretrained policy, which may be particularly advantageous for policy networks that are significantly larger than the MLP used in our experiments.

Table 3: Average steady-state error (in m) from the hovering target across 8 rollouts. The errors of the two best-performing methods for each disturbance condition are highlighted.

Method	No disturbance	Small disturbance	Large disturbance
Ours State	0.015 ± 0.001	0.008 ± 0.002	0.105 ± 0.007
Ours	0.082 ± 0.009	0.099 ± 0.021	0.207 ± 0.041
Ours (Residual)	0.084 ± 0.014	0.111 ± 0.024	0.205 ± 0.048

Computational and Sample Efficiency Analysis. We analyze and compare the sample and computational efficiency of our approach against DATT for state-based tasks. All runtimes are recorded from running on an Nvidia RTX 4090 GPU (24 GB VRAM) with an Intel 14900KF CPU. For our approach, policy pretraining uses 300 epochs across 100 environments, which is equivalent to 4.5 million steps in total, and takes approximately *15 seconds*. Each online policy adaptation step runs for 30 epochs across 10 environments (or 45k steps) and takes about *1.5 seconds*. Empirically, we observed that good-performing policies can in fact be obtained using fewer epochs and environments, thanks to the low-variance first-order policy gradients from differentiable simulation. In comparison, DATT trains the policy for 20 million steps, which takes around *2 hours*, and requires no further training at runtime. For DATT, we also observed slower convergence to lower rewards when training with larger domain randomization, which is likely a result of the performance-generalization trade-off [47], possibly exacerbating the high variance in the policy gradient estimates. Our approach enables more efficient compute usage by simplifying initial policy training without domain randomization or curricula, and by supporting sample- and compute-efficient online adaptation to out-of-distribution scenarios where domain randomization fails to generalize.



(a) State-based hovering with added mass and wind.

(b) *Figure-8* tracking on the large quadrotor.

Figure 4: Real-world policy adaptation using our proposed approach in real-time. Vertical dashed lines indicate policy update steps. Shaded regions represent the error from the reference.

4.3 Real-World Validation

We validated our approach in real-world experiments for both stabilizing hover and trajectory tracking tasks with various disturbance conditions, including both modifications to the hardware and wind disturbances. Detailed visualizations and experiments are provided in the supplementary materials. We used a motion-capture system to provide state estimation of the quadrotor at 100 Hz. The exact same procedures as the simulated experiments were used for both the pretraining and online adaptation phases. We use two quadrotors adapted from the Agilicious platform [11]: a small lightweight quadrotor and a larger, heavier one with different dynamical properties; further details are provided in Appendix D. An additional quadrotor stand was rigidly attached to the small quadrotor from below, increasing its mass from 120 g to 190 g by approximately 60% while simultaneously altering its inertial properties. Therefore, both the existing sim-to-real gap and the extra disturbances contribute to significant out-of-distribution dynamics that were unseen during policy pretraining. Fig. 4a shows the recorded trajectory of adaptation of the state-based hovering policy on the modified small quadrotor under a diagonal wind disturbance. Due to the highly imbalanced and non-uniform drag profile of the modified quadrotor, the wind disturbance forces are more complex and unstable than the constant, uniform disturbance used in simulated experiments. However, our method still enables the policy to rapidly adapt with 2-3 policy update steps to compensate for disturbances, which takes approximately 10 to 15 seconds. Similar results were observed for vision-based hovering; the adaptation process appeared less stable than state-based hovering, which is consistent with our findings from the simulated experiments. Real-world experiments show that our approach achieves accurate trajectory tracking under various disturbance conditions. Here, we present one particular experiment where a base *Figure-8*-tracking policy was deployed on the large quadrotor. As shown in Fig. 4b, despite the poor initial tracking and state-space exploration caused by the large sim-to-real gap, the policy quickly adapts and achieves much improved tracking within just a few policy update steps.

5 Discussions

We propose a novel rapid policy adaptation framework for combining online residual dynamics learning from real-world flight data and sample-efficient policy learning via differentiable simulation. With all system components designed for rapid adaptation, we demonstrate the possibility to adapt both state and vision-based policies to unknown disturbances within *several seconds*. One limitation is that our residual dynamics network only predicts residual linear accelerations. One limitation is that the proposed residual network does not model dynamic disturbances, i.e., disturbances with internal states. In order to improve the current architecture, recurrent networks or state-to-state residuals would model a larger class of time-dependent disturbances [26]. Moreover, due to the tightly-coupled dependencies between data collection via policy rollout and policy learning using learned residual dynamics from the collected data, the quality and rate of convergence may be affected by biases or noise in the learned residual dynamics. The dependency is closely related to the concept of *performative prediction* [48] in related machine learning fields. Thus, future work will explore uncertainty-driven data collection where the policy is augmented by active exploration to simultaneously improve task performance and reduce uncertainty in the real-world dynamics.

References

[1] J. Collins, S. Chand, A. Vanderkop, and D. Howard. A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431, 2021.

[2] A. Waheed, M. Areti, L. Gallantree, and Z. Hasnain. Quantifying the sim2real gap: Model-based verification and validation in autonomous ground systems. *IEEE Robotics and Automation Letters*, 2025.

[3] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[4] J. Tobin, L. Biewald, R. Duan, M. Andrychowicz, A. Handa, V. Kumar, B. McGrew, A. Ray, J. Schneider, P. Welinder, et al. Domain randomization and generative models for robotic grasping. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3482–3489. IEEE, 2018.

[5] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[6] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. Solving rubik's cube with a robot hand, 2019. URL https://arxiv.org/abs/1910.07113.

[7] H. Wang, J. Xing, N. Messikommer, and D. Scaramuzza. Environment as policy: Learning to race in unseen tracks. *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[8] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull. Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176. PMLR, 2020.

[9] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, Aug 2023. ISSN 1476-4687.

[10] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[11] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, et al. Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. *Science Robotics*, 7(67):eabl6259, 2022.

[12] K. Huang, R. Rana, A. Spitzer, G. Shi, and B. Boots. Datt: Deep adaptive trajectory tracking for quadrotor control. In *Conference on Robot Learning*, pages 326–340. PMLR, 2023.

[13] R. Newbury, J. Collins, K. He, J. Pan, I. Posner, D. Howard, and A. Cosgun. A review of differentiable simulators. *IEEE Access*, 2024.

[14] J. Heeg, Y. Song, and D. Scaramuzza. Learning quadrotor control from visual features using differentiable simulation. In *2025 International Conference on Robotics and Automation (ICRA)*. IEEE, 2025.

[15] E. Nava, J. Z. Zhang, M. Y. Michelis, T. Du, P. Ma, B. F. Grewe, W. Matusik, and R. K. Katzschmann. Fast aquatic swimmer optimization with differentiable projective dynamics and neural network hydrodynamic models. In *International Conference on Machine Learning*, pages 16413–16427. PMLR, 2022.

[16] E. Heiden, M. Macklin, Y. Narang, D. Fox, A. Garg, and F. Ramos. Disect: A differentiable simulation engine for autonomous robotic cutting. In *Robotics: Science and Systems*, 2021.

[17] J. Degrave, M. Hermans, J. Dambre, and F. Wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13:6, 2019.

[18] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros. Add: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[19] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin. Scalable differentiable physics for learning and control. In *Proceedings of the 37th International Conference on Machine Learning*, pages 7847–7856, 2020.

[20] J. Xu, S. Kim, T. Chen, A. R. Garcia, P. Agrawal, W. Matusik, and S. Sueda. Efficient tactile simulation with differentiability for robotic manipulation. In *Conference on Robot Learning*, pages 1488–1498. PMLR, 2023.

[21] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2022.

[22] I. Georgiev, K. Srinivasan, J. Xu, E. Heiden, and A. Garg. Adaptive horizon actor-critic for policy learning in contact-rich differentiable simulation. In *Proceedings of the 41st International Conference on Machine Learning*, pages 15418–15437, 2024.

[23] J. Y. Luo, Y. Song, V. Klemm, F. Shi, D. Scaramuzza, and M. Hutter. Residual policy learning for perceptive quadruped control using differentiable simulation. In *2025 International Conference on Robotics and Automation (ICRA)*. IEEE, 2025.

[24] L. Ljung. *System Identification (2nd ed.): Theory for the User*. Prentice Hall PTR, USA, 1999. ISBN 0136566952.

[25] A. Chiuso and G. Pillonetto. System identification: A machine learning perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):281–304, 2019.

[26] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme. Neuralsim: Augmenting differentiable simulators with neural networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9474–9481. IEEE, 2021.

[27] G. Cioffi, L. Bauersfeld, and D. Scaramuzza. Hdvio: Improving localization and disturbance estimation with hybrid dynamics vio. In *Robotics: Science and Systems*, 2023.

[28] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza. Neurobem: Hybrid aerodynamic quadrotor model. In *Robotics: Science and Systems*, 2024.

[29] G. Cioffi, L. Bauersfeld, and D. Scaramuzza. Hdvio2.0: Wind and disturbance estimation with hybrid dynamics vio. *arXiv preprint arXiv:2504.00969*, 2025.

[30] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza. Data-driven mpc for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):3769–3776, 2021.

[31] J. Gao, M. Y. Michelis, A. Spielberg, and R. K. Katzschmann. Sim-to-real of soft robots with learned residual physics. *IEEE Robotics and Automation Letters*, 2024.

[32] J. Xing, L. Bauersfeld, Y. Song, C. Xing, and D. Scaramuzza. Contrastive learning for enhancing robust scene transfer in vision-based agile flight. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

[33] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. Neural lander: Stable drone landing control using learned dynamics. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9784–9790. IEEE, 2019.

[34] S. Chen, K. Werling, A. Wu, and C. K. Liu. Real-time model predictive control and system identification using differentiable simulation. *IEEE Robotics and Automation Letters*, 8(1): 312–319, 2022.

[35] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. Neural-fly enables rapid learning for agile flight in strong winds. *Science Robotics*, 7(66): eabm6597, 2022.

[36] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. In *Robotics: Science and Systems*, 2021.

[37] J. Xing, I. Geles, Y. Song, E. Aljalbout, and D. Scaramuzza. Multi-task reinforcement learning for quadrotors. In *IEEE Robotics and Automation Letters (RA-L)*. IEEE, 2024.

[38] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

[39] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

[40] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] Y. Song, S. bae Kim, and D. Scaramuzza. Learning quadruped locomotion using differentiable simulation. In *8th Annual Conference on Robot Learning*, 2024.

[42] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[43] J. Xing, A. Romero, L. Bauersfeld, and D. Scaramuzza. Bootstrapping reinforcement learning with imitation for vision-based agile flight. In *8th Annual Conference on Robot Learning*, 2024.

[44] R. Trumpp, E. Javanmardi, J. Nakazato, M. Tsukada, and M. Caccamo. Racemop: Mapless online path planning for multi-agent autonomous racing using residual policy learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8449–8456. IEEE, 2024.

[45] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*, 2024.

[46] K. Huang, R. Rana, A. Spitzer, G. Shi, and B. Boots. Datt: Deep adaptive trajectory tracking for quadrotor control. https://github.com/KevinHuang8/DATT, 2024.

[47] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy. Domain generalization: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4396–4415, 2022.

[48] J. Perdomo, T. Zrnic, C. Mendler-Dünner, and M. Hardt. Performative prediction. In *International Conference on Machine Learning*, pages 7599–7609. PMLR, 2020.

A Low-Fidelity Quadrotor Dynamics Model

We model the quadrotor as a point mass, and use $\boldsymbol{p} \in \mathbb{R}^3$, $\boldsymbol{R} \in \mathbb{SO}(3)$ and $\boldsymbol{v} \in \mathbb{R}^3$ to denote its position, orientation matrix, and linear velocity in the global frame, respectively. Furthermore, let $c \in \mathbb{R}$ and $\boldsymbol{\omega} \in \mathbb{R}^3$ represent the commanded collective thrust and body rates (angular velocity) respectively. The low-fidelity quadrotor dynamics is given by:

$$\dot{\boldsymbol{x}} = \frac{d}{dt}\begin{bmatrix} \boldsymbol{p} \\ \mathrm{vec}(\mathrm{R}) \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \mathrm{vec}(\mathrm{R}[\boldsymbol{\omega}]_\times) \\ \boldsymbol{R}\boldsymbol{c} + \boldsymbol{g} \end{bmatrix} \tag{3}$$

where $[\cdot]_\times$ denotes the skew-symmetric matrix operator and $\mathrm{vec}(\cdot)$ indicates vectorization of a matrix, $\boldsymbol{c} = [0, 0, c]^\top$ is the collective thrust vector, and $\boldsymbol{g}$ is the gravity vector.

B Reward Definitions for Evaluation Tasks

For both the stabilizing hover and trajectory tracking tasks, the reward at each time step t is defined as a sum of position, velocity, and actuation reward terms:

$$r_t = r_t^{\mathrm{pos}} + r_t^{\mathrm{vel}} + r_t^{\mathrm{act}} \tag{4}$$

For stabilizing hover, the individual reward terms are given by:

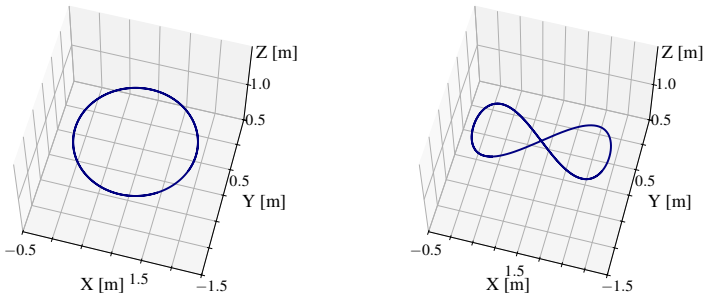
$$\begin{aligned} r_t^{\mathrm{pos}} &= -1.0 \cdot L_{\mathrm{H}}(5 \cdot (\boldsymbol{p}_t - \boldsymbol{p}_{\mathrm{des}})) \\ r_t^{\mathrm{vel}} &= -0.1 \cdot L_{\mathrm{H}}(\boldsymbol{v}_t) - 0.1 \cdot L_{\mathrm{H}}(\boldsymbol{\omega}_t) \\ r_t^{\mathrm{act}} &= -0.5 \cdot L_{\mathrm{H}}(\boldsymbol{u}_t - \boldsymbol{u}_{\mathrm{hover}}) \end{aligned} \tag{5}$$

where L_{H} is the Huber loss, and $\boldsymbol{u}_{\mathrm{hover}} = [9.81, 0, 0, 0]^\top$ is the mass-normalized action required to counteract gravity.

For trajectory tracking, the individual reward terms are given by:

$$\begin{aligned} r_t^{\mathrm{pos}} &= -1.0 \cdot L_{\mathrm{H}}(\boldsymbol{p}_t - \boldsymbol{p}_t^{\mathrm{ref}}) \\ r_t^{\mathrm{vel}} &= -1.0 \cdot L_{\mathrm{H}}(\boldsymbol{v}_t - \boldsymbol{v}_t^{\mathrm{ref}}) \\ r_t^{\mathrm{act}} &= -0.1 \cdot L_{\mathrm{H}}(\boldsymbol{u}_t - \boldsymbol{u}_{\mathrm{hover}}) \end{aligned} \tag{6}$$

Here, $\boldsymbol{p}_t^{\mathrm{ref}}$ and $\boldsymbol{v}_t^{\mathrm{ref}}$ are respectively the corresponding position and velocity of the reference trajectory at time t. As shown in Fig. 5, both the *Circle* and *Figure-8* trajectories lie in the horizontal xy-plane at a height of 1 m above the ground. The *Circle* trajectory has a radius of 1 m and a period of 3 seconds. The *Figure-8* trajectory spans 3 m and 1 m across the x and y directions respectively, and has a period of 5 seconds. Both trajectories start at the point $(0, 0, 1)$ m.

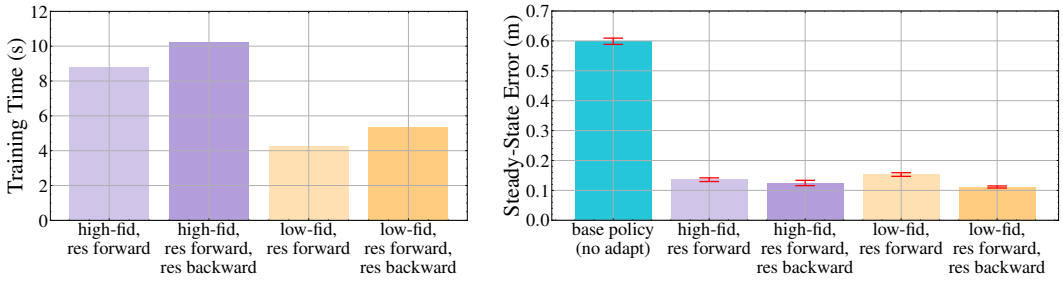


(a) *Circle* (b) *Figure-8*

Figure 5: The *Circle* and *Figure-8* reference trajectories, with periods of 3 seconds and 5 seconds respectively.

C Optimizing Runtime Efficiency and Performance

We conducted an analysis using the state-based stabilizing hover task to justify two key design choices in the differentiable simulation pipeline: 1) low-fidelity analytical dynamics model for simulation, and 2) gradient backpropagation only through the analytical dynamics model. Given that a key objective is to minimize runtime while maintaining policy performance, we compared the training time and policy performance for each design choice. We used the same realistic quadrotor simulator [11] and added a constant uniform acceleration disturbance of $2\,\mathrm{m/s^2}$ in the positive x-axis direction. We first collected 50 rollout trajectories (3 seconds each) of the base hovering policy from 50 random starting configurations around the hovering target, and then used the generated residual samples to train a single residual dynamics network for 200 epochs. Finally, we adapted the base policy by running 100 epochs of policy training across 100 parallel environments, and evaluated the final steady-state errors from the hovering target across 8 rollout trajectories.



(a) Training time

(b) Final steady-state error

Figure 6: (left) Policy training times using four different simulation configurations. (right) Resulting policy performances compared against the base policy performance (no adaptation). Error bars show ± 3 standard deviations of the error distribution across 8 rollouts for each configuration.

We first compared using the low-fidelity (low-fid) dynamics model (3) against using a high-fidelity model (high-fid) which simulates body and rotor drag effects, the rotor thrust maps, and the low-level controller dynamics, as the analytical dynamics model together with the residual dynamics network for forward simulation (res forward). We found that using the low-fidelity model achieves approximately 2-times faster training (see Fig. 6a) than using the high-fidelity model, while the achieved policy performances by both methods were very comparable (see Fig. 6b). For gradient backpropagation, we found that in addition to backpropagating through the analytical dynamics model, also performing backpropagation through the learned residual dynamics network (res backward) increases training time by approximately 30% without providing clear benefits to the policy performance. This is consistent with previous findings [14, 41] that combining accurate forward simulation with a surrogate gradient which points in approximately the same direction as the true gradient vector accelerates policy training without impacting the resulting policy performance.

D Quadrotor Platform

In this section, we list the physical properties of the quadrotor platforms used in our experiments.

Table 4: Overview of the quadrotor parameters for both simulation and real-world experiments.

Param.	**Small Quadrotor**	**Large Quadrotor**
Mass [kg]	0.12	0.60
Maximum Thrust [N]	14.00	34.00
Arm Length [m]	0.06	0.13
Inertia [$\mathrm{g\,m^2}$]	[0.14, 0.17, 0.21]	[2.41, 1.80, 3.76]