Learning on the Fly: Rapid Policy Adaptation via Differentiable Simulation

Jiahe Pan*, Jiaxu Xing*, Rudolf Reiter, Yifan Zhai, Elie Aljalbout, and Davide Scaramuzza

***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 can degrade policy performance. Existing approaches, such as domain randomization and Real2Sim2Real pipelines, can improve policy robustness, but either struggle under out-of-distribution conditions or require costly offline retraining. In this work, we approach these problems from a different perspective. Instead of relying on diverse training conditions before deployment, we focus on rapidly adapting the learned policy in the real world in an online fashion. To achieve this, we propose a novel online adaptive learning framework that unifies residual dynamics learning with real-time policy adaptation inside a differentiable simulation. Starting from a simple dynamics model, our framework continuously refines the model using real-world data to capture unmodeled effects and disturbances, such as payload changes and wind. The refined dynamics model is embedded in a differentiable simulation framework, enabling gradient backpropagation through the dynamics and thus rapid, sample-efficient policy updates beyond the reach of classical RL methods like PPO. All components of our system are designed for rapid adaptation, enabling the policy to adjust to unseen disturbances within *5 seconds* of training. We validate the approach on agile quadrotor control under various disturbances in both simulation and the real world. Our framework reduces hovering error by up to 81% compared to $\mathcal{L}_1$-MPC and 55% compared to DATT, while also demonstrating robustness in vision-based control without explicit state estimation.**

Supplementary Material

Video: https://youtu.be/euK2GbcNTvk

Website: https://rpg.ifi.uzh.ch/lotf/

Code: https://github.com/uzh-rpg/learning_on_the_fly

I. 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. While a high-fidelity simulation model may be used, the system parameters are often difficult to precisely identify. In addition, unmodeled effects such as aerodynamic turbulence, sensor noise, and actuator delays further complicate the real-world dynamics, thus making accurate alignment between simulation and reality difficult to achieve. 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 real-world complexity and variability.

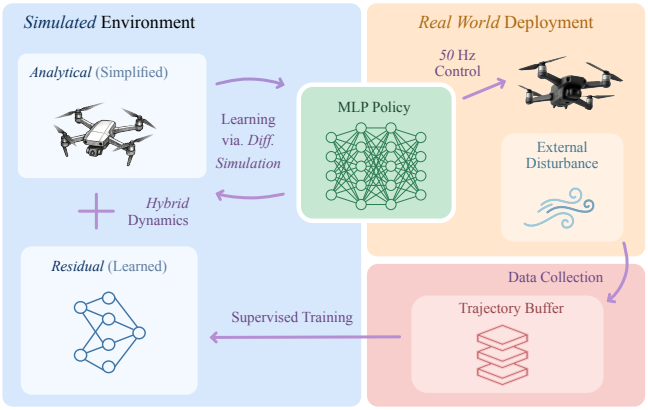


Fig. 1: Overview of the key components in our proposed approach. (right) The policy is continuously deployed in the real world, and trajectories are collected into a buffer. (bottom) Residual dynamics are trained using the real-world data to refine the *hybrid* simulation dynamics. (left) Based on the latest simulation dynamics, the policy is rapidly adapted via differentiable simulation.

Domain randomization is a common strategy to address this issue [3], in which simulation parameters such as dynamics [4], [5], sensor noise [6], [7], and visual appearance [3], [8] of the environment 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 exhaustively anticipate all possible real-world conditions. Thus, when the environment shifts beyond the randomized distribution, policy performance will strongly degrade to out-of-distribution disturbances [9]. Beyond domain randomization, Real2Sim2Real methods [6], [5] have shown strong sim-to-real transfer capabilities through offline refinement of the simulation model using real-world data before retraining policies for deployment. While such methods are effective in improving transfer, they require extensive data and costly retraining. For example, [10] reports 75 minutes of real-world data collection, which makes them unsuitable for rapid online adaptation to changing conditions.

In this work, we approach these problems from another perspective: we propose to *rapidly adapt the policy* to unknown external disturbances in the real world, in an online fashion. The core insight is to integrate online residual

* indicates equal contribution. All authors are with the Robotics and Perception Group, Department of Informatics, University of Zurich, Switzerland (https://rpg.ifi.uzh.ch).

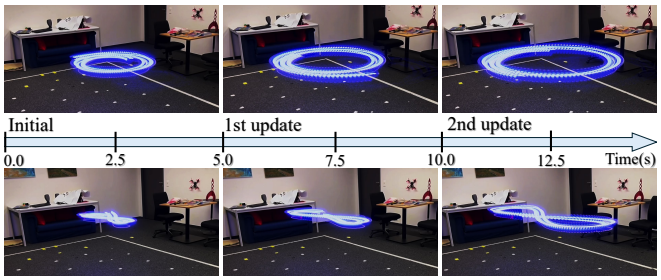


Fig. 2: Real-world trajectory tracking adaptation using our proposed approach. The policy rapidly learns within 2 updates (10 s of flight) to compensate for the large sim-to-real gap caused by model mismatch in pretraining (see Sec. III).

dynamics learning with rapid policy adaptation via differentiable simulation. All system components are designed to update both the residual dynamics and policy *as quickly as possible*, ideally within a few seconds, during runtime. In this way, the policy becomes adaptive by continually "overfitting" rapidly to the current environment scenario, and paradoxically, more "generalizable" across diverse conditions.

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. The residual-augmented dynamics model is embedded in a differentiable simulation framework to achieve accurate and sample-efficient policy adaptation. Differentiable simulation provides the key advantage here: by allowing accurate gradients to flow through the dynamics, it makes real-world policy adaptation more efficient than classical RL approaches such as PPO [11]. 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 through simulation using the refined dynamics. All components ensure the simulation is aligned with reality and enables rapid, efficient policy adaptation to unknown disturbances, even controlling directly from perceptual input.

We evaluate our proposed framework in both simulation and real-world experiments across various environmental disturbance conditions on an agile quadrotor platform, whose nonlinear dynamics and sensitivity to aerodynamic effects make it a challenging benchmark for adaptive control [12]. In state-based control tasks such as hovering, where the policy receives as input full quadrotor state information, our method attains an average error of $0.105\,\mathrm{m}$, an 81% reduction over $\mathcal{L}_1$-MPC ($0.552\,\mathrm{m}$) and 55% over DATT [13] ($0.231\,\mathrm{m}$), while ensuring stable flight under modeling errors and out-of-distribution disturbances. In visual feature-based control, our framework achieves similar gains, demonstrating that rapid adaptation remains effective under partial or noisy observations - a capability unattainable with classical control methods in the absence of state estimation.

II. Related Works

A. 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 [14], but its representation capacity is limited to the modeled system [15], making it less effective for capturing complex dynamics and disturbances. 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 [10], [16], learn motor delays in quadrupeds [6], and predict residual forces in soft robots [17].

B. Fast Policy Learning in Simulation

While traditional RL methods suffer from prohibitively long training times, significant speed-ups can be achieved using highly optimized physics simulation [18]. However, these methods are limited by sample inefficiency due to the high variance in the zeroth-order policy gradient estimates. A recent alternative paradigm, policy learning via differentiable simulation, uses smooth, differentiable dynamics and rewards to enable policy learning via *first-order* gradients [19], offering substantial gains in sample efficiency and training time over RL [20]. It has been applied to direct policy parameterizations, such as parametric curve frequencies for swimming robots [21] and sinusoidal policies for robotic cutting [22]. For applications to neural network policies, however, unstable gradients often limit applications to short-horizon tasks with simplified contacts and restricted start-state variation [23]. To address this, prior work has explored enhancements such as early-stopping simulations at contact events, truncated BPTT [24], and reward augmentation with a learned critic [25], [26].

C. Learning-Based Adaptive Control

Residual dynamics models have been used for online disturbance estimation via offline-trained networks [27], Gaussian Processes [16], or differentiable simulation–based system identification [28]. However, 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 network policies have also been conditioned on disturbance estimates [13], [29], [4], but since they are trained offline in randomized simulations and remain fixed during deployment, they struggle with domain shifts and unseen real-world conditions [30], [13].

III. Learning on the Fly

Our approach consists of two phases: policy pretraining and online adaptation. During pretraining, we train a base policy for online adaptation using a low-fidelity analytical dynamics model *without* residual dynamics. During online adaptation (see Fig. 3), residual dynamics learning, policy adaptation, and real-world deployment run in parallel across multiple threads, with parameters exchanged efficiently between processes via ROS as serialized byte strings. The residual dynamics network is continuously updated from a rolling buffer of quadrotor states and actions, and combined with the analytical model to form a hybrid dynamics embedded in the differentiable simulation for policy adaptation. The

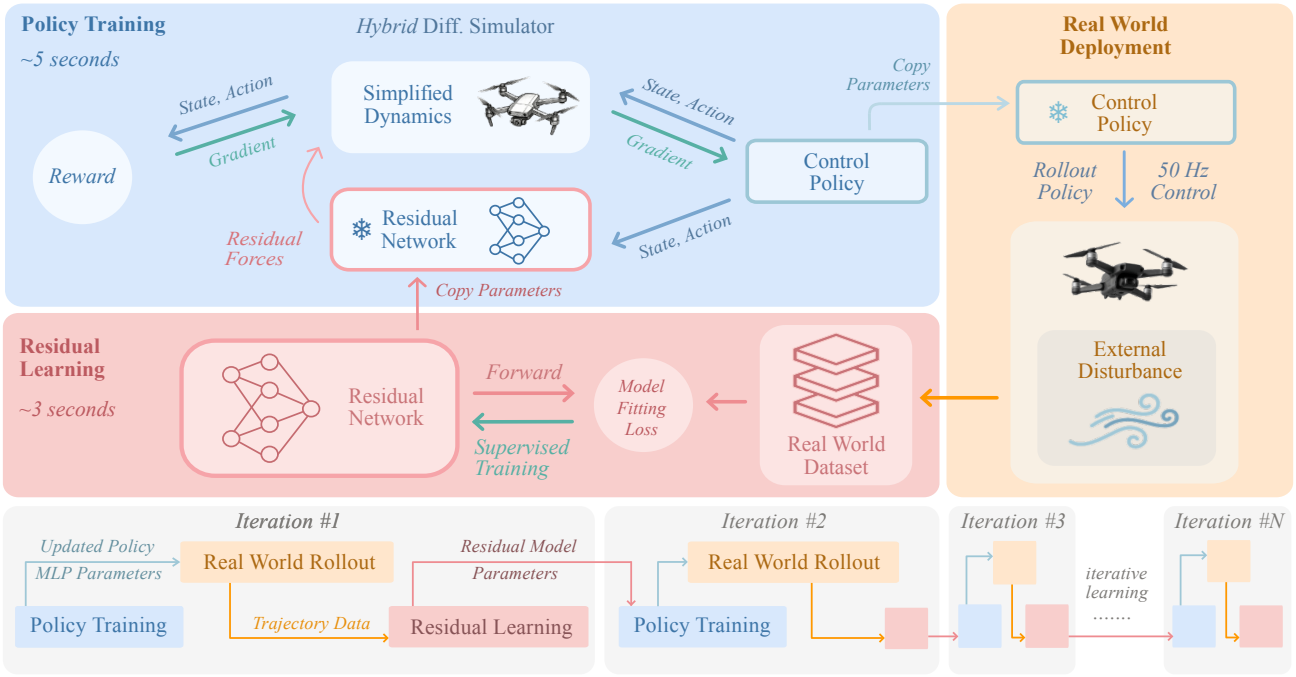


Fig. 3: Detailed illustration of the information flow within and between the three interleaved components of the proposed framework, including residual dynamics learning, differentiable simulation, and policy adaptation. These components operate concurrently across multiple threads in separate ROS nodes.

deployment loop uses the latest policy network parameters to continuously output control commands at 50 Hz to the on-board flight controller, and simultaneously collects flight trajectories.

A. Differentiable Simulation Model

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 differentiable 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. Thus, all components are fully differentiable and allow gradient backpropagation through the simulation.

B. Low-Fidelity Quadrotor Dynamics Model

Given the quadrotor state $\boldsymbol{x}$ consisting of position $\boldsymbol{p} \in \mathbb{R}^3$, rotation matrix $\boldsymbol{R} \in \mathrm{SO}(3)$, and linear velocity $\boldsymbol{v} \in \mathbb{R}^3$, and commands $\boldsymbol{u}$ consisting of the mass-normalized collective thrust $c \in \mathbb{R}$ and body rates $\boldsymbol{\omega}_{\text{cmd}} \in \mathbb{R}^3$, the low-fidelity, analytical quadrotor dynamics f_{a} is defined as

$$\dot{\boldsymbol{x}} = \frac{d}{dt}\begin{bmatrix} \boldsymbol{p} \\ \mathrm{vec}(\boldsymbol{R}) \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \mathrm{vec}(\boldsymbol{R}[\boldsymbol{\omega}_{\text{cmd}}]_\times) \\ \boldsymbol{R}\boldsymbol{c} + \boldsymbol{g} \end{bmatrix} \coloneqq f_{\text{a}}(\boldsymbol{x}, \boldsymbol{u}), \quad (1)$$

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.

C. Policy Optimization Using Analytical Gradients

The policy learning objective is to maximize the cumulative task reward $\mathcal{R}(\phi)$ over an N-step rollout of the policy parameterized by ϕ via

$$\max_{\phi} \mathcal{R}(\phi) = \sum_{t=0}^{N-1} r(x_t, u_t) = \sum_{t=0}^{N-1} r(x_t, \pi_\phi(h(x_t))). \quad (2)$$

By leveraging the differentiable dynamics and reward structure, we can obtain first-order analytical policy gradients of the objective (2) via Back-Propagation Through Time (BPTT) (see [31] for a full derivation). The gradient and the update rule of the policy parameters ϕ are given by

$$\nabla_\phi R(\phi) = \frac{1}{N}\sum_{t=0}^{N-1}\Big(\sum_{i=1}^{t}\frac{\partial r_t}{\partial x_t}\prod_{j=1}^{t}\Big(\frac{dx_j}{dx_{j-1}}\Big)\frac{\partial x_i}{\partial \phi} + \frac{\partial r_t}{\partial u_t}\frac{\partial u_t}{\partial \phi}\Big),$$
$$\phi_{k+1} = \phi_k + \alpha \nabla_\phi R(\phi_k), \quad (3)$$

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 [20] written entirely in JAX to leverage both its automatic-differentiation framework for computing the analytical policy gradients and performing GPU-accelerated parallel simulation.

D. Residual Dynamics Learning

Given the concatenated input vector $[\boldsymbol{x}^\top, \boldsymbol{u}^\top] \in \mathbb{R}^{19}$ of quadrotor state $\boldsymbol{x}^\top = [\boldsymbol{p}^\top, \mathrm{vec}(\boldsymbol{R})^\top, \boldsymbol{v}^\top] \in \mathbb{R}^{15}$ and action $\boldsymbol{u}^\top = [c, \boldsymbol{\omega}_{\text{cmd}}^\top] \in \mathbb{R}^4$, an MLP network f_{res} parameterized by $\boldsymbol{\theta}$ 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 $\hat{\boldsymbol{a}} \in \mathbb{R}^3$ from the analytical dynamics

$f_a(\boldsymbol{x}, \boldsymbol{u})$ in (1). 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 $\{[\boldsymbol{x}^\top, \boldsymbol{u}^\top]^i, \boldsymbol{a}^i_{\text{res}}\}_{i \in \mathcal{B}}$, we train the model by minimizing the loss function $\mathcal{L}_{\text{res}}$ via $\min_\theta \mathcal{L}_{\text{res}} = \min_\theta \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} ||\boldsymbol{a}^i_{\text{res}} - f_{\text{res}}([\boldsymbol{x}^\top, \boldsymbol{u}^\top]^i; \boldsymbol{\theta})||^2 + \beta \sum_{l=1}^{L} ||W^l||_2^2$, where W^l is the weight matrix of the l-th network layer, and β controls the regularization strength. 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 [32] by regulating the network's Lipschitz constant [27].

E. 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} dynamics models. Here, we use a simple, low-fidelity analytical dynamics model (see Sec. III-B) 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. The quadrotor states are simulated at 50 Hz via Runge-Kutta 4 time-integration of the dynamics 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 inspired by prior work in policy learning using differentiable simulation for both quadruped [11] and quadrotor [20] control, which showed that combining accurate forward dynamics simulation with the backpropagation of a surrogate gradient based on a simplified 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 Sec. IV-C.

F. Full vs. Low-Rank Policy Adaptation

We compare two existing methods of adapting a pretrained policy: full vs. low-rank adaptation (LoRA) [33]. Full adaptation involves updating all parameters of the policy network, similar to [34], whereas LoRA freezes all pretrained parameters and instead adapts an additive low-rank network module [35] which forms a lower-dimensional trainable parameter space. The latter has been shown to achieve more memory and parameter-efficient policy adaptation using RL for task-transfer [33] and multi-agent [36] learning. Therefore, as an exploratory comparison, we seek to understand whether LoRA can also be combined with sample-efficient policy learning using differentiable simulation to effectively adapt a pretrained policy to unknown environmental disturbances.

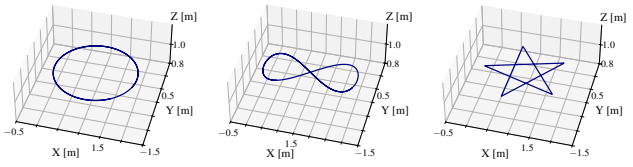
IV. Experiments

A. Experimental Setup

1) Task and Reward Definitions: We evaluate our approach on quadrotor stabilizing hover and trajectory tracking. For stabilizing hover, the policy is required to regulate the quadrotor towards a goal $\boldsymbol{p}_{\text{des}}$ and maintain it at all times, which is non-trivial given the quadrotor's non-linear and unstable dynamics. We evaluate both a state-based policy which receives observations $\boldsymbol{o} = [\boldsymbol{p}, \boldsymbol{R}, \boldsymbol{v}]^\top$ at each time step, and an end-to-end visual feature-based policy which only receives the projected pixel coordinates of seven 3D keypoints from the past five time steps and the last three actions. For real-world experiments, the 3D keypoints are simulated in a hardware-in-the-loop style using quadrotor state estimates from a motion-capture system. Our training setup closely follows the open-source environment setup in [20]. Trajectory tracking requires following a reference trajectory defined as a time-parameterized sequence of quadrotor states, with policy training done in a state-based setting. As shown in Fig. 4, we generate two smooth trajectories, *Circle* and *Figure-8*, and a non-smooth *5-Point Star* trajectory featuring a highly discontinuous velocity profile. For both tasks, the reward at each time step t is defined as a sum of position, velocity, and actuation rewards $r_t = r_t^{\text{pos}} + r_t^{\text{vel}} + r_t^{\text{act}}$. For stabilizing hover, the individual reward terms are $r_t^{\text{pos}} = -1.0 \cdot L_{\text{H}}(5 \cdot (\boldsymbol{p}_t - \boldsymbol{p}_{\text{des}}))$, $r_t^{\text{vel}} = -0.1 \cdot L_{\text{H}}(\boldsymbol{v}_t) - 0.1 \cdot L_{\text{H}}(\boldsymbol{\omega}_t)$, and $r_t^{\text{act}} = -0.5 \cdot L_{\text{H}}(\boldsymbol{u}_t - \boldsymbol{u}_{\text{hover}})$, where L_{H} is the Huber loss, and $\boldsymbol{u}_{\text{hover}} = [9.81, 0, 0, 0]^\top$ is the mass-normalized action required to counteract gravity. For trajectory tracking, the individual reward terms are $r_t^{\text{pos}} = -1.0 \cdot L_{\text{H}}(\boldsymbol{p}_t - \boldsymbol{p}_t^{\text{ref}})$, $r_t^{\text{vel}} = -1.0 \cdot L_{\text{H}}(\boldsymbol{v}_t - \boldsymbol{v}_t^{\text{ref}})$, and $r_t^{\text{act}} = -0.1 \cdot L_{\text{H}}(\boldsymbol{u}_t - \boldsymbol{u}_{\text{hover}})$, where $\boldsymbol{p}_t^{\text{ref}}$ and $\boldsymbol{v}_t^{\text{ref}}$ are respectively the reference position and velocity at time t.

2) 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. For visual feature-based hovering, we use the initialization approach from [20] to first train a neural network on a state-representation learning task and use the learned parameters to partially initialize the policy network. We refer the reader to [20] for details on this initialization method. We then train the partially initialized policy for 500 epochs across 300 parallel environments.

3) 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 stabilizing hover and trajectory tracking, we use history buffer sizes of 100 and 250, equivalent to 2 and 5 seconds of trajectory history, respectively. For residual dynamics learning, we continuously refine an ensemble of 3 networks,



(a) *Circle* (b) *Figure-8* (c) *5-Point Star*

Fig. 4: The *Circle*, *Figure-8* and *5-Point Star* reference trajectories, with periods of 3 s, 5 s and 6 s respectively. All trajectories lie in the horizontal xy-plane 1 m above the ground, and start at the point $(0, 0, 1)$ m.

TABLE I: 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 in green and orange.

Method	No Dist.	Small Dist.	Large Dist.
Base DiffSim	0.128 ± 0.004	0.328 ± 0.001	1.228 ± 0.073
$\mathcal{L}_1$-MPC	0.091 ± 0.052	0.134 ± 0.073	0.552 ± 0.130
DATT (PPO)	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 (LoRA)	0.023 ± 0.002	0.015 ± 0.004	0.125 ± 0.002

each with two 128-dim hidden layers and initialized using 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. For LoRA, we follow the original implementation and initialization in [37], and use ranks of 4, 10, and 1 for the three weight matrix layers of the policy network, respectively, and a constant scale of 4 across all layers. We run residual dynamics learning every 3 seconds and train the ensemble networks in parallel for 100 iterations. Policy adaptation is run every 5 seconds, and we train the state-based policy with 10 parallel simulated environments for 30 epochs and the vision-based policy with 30 environments for 50 epochs. These values were empirically found to provide a good balance between training time and policy performance.

4) Baselines Methods: We compare against a state-of-the-art learning-based adaptive control method, Deep Adaptive Tracking Control (DATT) [13], which uses the popular model-free RL algorithm PPO 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) [4], which is a similar approach but instead uses a learned encoder for disturbance estimation. We used the open-source implementation of [13] and the exact same training procedure and hyperparameters to train both state-based hovering and trajectory tracking policies using PPO for 20 million simulation steps. Using their original domain randomization method, 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 [13], which uses a Model Predictive Path Integral (MPPI) formulation and the same $\mathcal{L}_1$ adaptive control-based disturbance estimation as in DATT. Additionally, we include our pretrained base policy (Base DiffSim) without online adaptation for comparison.

B. Experimental Results

We used a realistic quadrotor simulator [12] equipped with the BEM model for aerodynamic effects and high-frequency simulation of 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.

TABLE II: Average tracking errors (in m) for three different trajectories (*Circle*, *Figure-8*, and *5-Point Star*). The errors of the two best-performing methods for each disturbance condition are highlighted in green and orange.

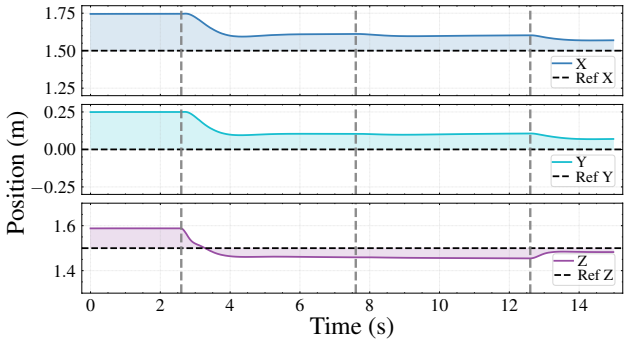
Trajectory	Method	No Dist.	Small Dist.	Large Dist.
Circle	Base DiffSim	0.365 ± 0.124	0.571 ± 0.091	1.479 ± 0.213
	$\mathcal{L}_1$-MPC	0.113 ± 0.027	0.096 ± 0.063	0.410 ± 0.155
	DATT (PPO)	0.058 ± 0.016	0.040 ± 0.024	*crash*
	Ours	0.167 ± 0.048	0.135 ± 0.101	0.349 ± 0.175
	Ours (LoRA)	0.129 ± 0.051	0.159 ± 0.043	0.326 ± 0.061
Figure-8	Base DiffSim	0.313 ± 0.087	0.492 ± 0.155	1.363 ± 0.382
	$\mathcal{L}_1$-MPC	0.109 ± 0.063	0.121 ± 0.025	0.281 ± 0.097
	DATT (PPO)	0.078 ± 0.037	0.082 ± 0.046	*crash*
	Ours	0.068 ± 0.040	0.045 ± 0.03	0.137 ± 0.098
	Ours (LoRA)	0.069 ± 0.047	0.059 ± 0.043	0.110 ± 0.037
5-Point Star	Base DiffSim	0.453 ± 0.190	0.467 ± 0.236	0.844 ± 0.389
	$\mathcal{L}_1$-MPC	0.295 ± 0.086	0.218 ± 0.111	0.417 ± 0.086
	DATT (PPO)	0.087 ± 0.059	0.102 ± 0.093	*crash*
	Ours	0.126 ± 0.094	0.133 ± 0.075	0.211 ± 0.116
	Ours (LoRA)	0.129 ± 0.069	0.130 ± 0.099	0.231 ± 0.076

All experiments (simulated and real-world) were run using an Nvidia RTX 4090 GPU (24 GB VRAM) with an Intel 14900KF CPU.

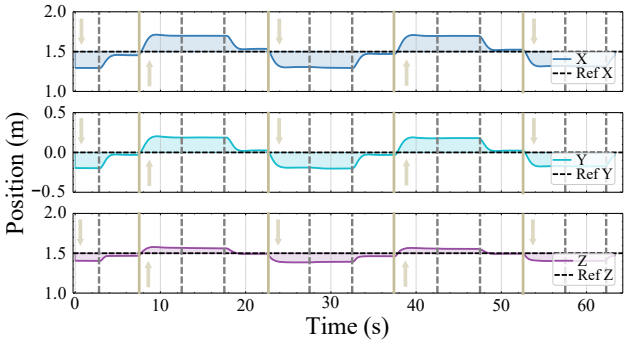
1) Performance Comparison to Baseline Approaches: For the state-based stabilizing hover task, we ran each method under all disturbance conditions from a set of 8 different starting positions around the hovering target, and used the final steady-state error as the performance metric. To ensure a fair comparison, we continued running each method until no further accuracy improvements were observed. This was found to be approximately 10 seconds for all baseline methods, as they do not require any policy adaptation, and around 30 seconds for our approach (both state and vision-based) for a few learning steps to take place. As summarized in Tab. I, results show that our method consistently exhibits superior or comparable performance to the baselines. Fig. 5a illustrates that our method rapidly adapts the policy to compensate for the *large* disturbances within 2-3 adaptation steps. DATT performs well under both the *none* and *small* disturbance scenarios, which are within its training distribution, but struggles to adapt to the larger, out-of-distribution disturbance.

For visual feature-based hovering, the baseline methods would require an additional state estimation module, which limits the ability to clearly benchmark their performance without confounding errors from the state estimator, and are thus excluded from the comparison. As shown in Tab. III, our visual feature-based approach resulted in larger errors than our state-based approach. We empirically observed that adaptation of the visual feature-based policy is less stable than the state-based counterpart and may require more policy learning epochs or update steps, most likely due to partial state observability and sample inefficiency in learning vision-based control. More detailed experiments and performance results are provided in the supplementary material.

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. II, our approach achieves comparable performance to the baselines across all trajectories and disturbance conditions. Here, our method exhibits consistent responsiveness



(a) State-based hover adaptation to constant *large* disturbance.



(b) Continual state-based hover adaptation to *varying* disturbances.

Fig. 5: Online adaptation of a state-based hovering policy to constant (a) and time-varying (b) disturbances. Vertical dashed lines indicate policy update steps which occur every 5 s. Shaded regions represent the error from the hovering target. In (b), each pair of solid vertical line and arrow indicates a direction reversal of the disturbance and the new direction.

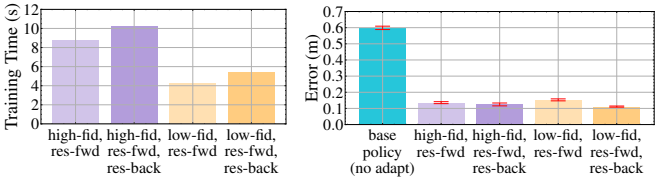
to both smooth and non-smooth references. In particular, for the *5-Point Star*, our method achieves comparable tracking accuracy as DATT, which has demonstrated strong capabilities in tracking non-smooth, dynamically infeasible trajectories [13]. We observed that our method is able to rapidly adapt and achieve much improved tracking accuracy after only 3-4 policy update steps. For DATT, consistent with findings from state-based hovering, it fails to generalize to the out-of-distribution disturbances and results in crashes for all reference trajectories.

Finally, we observe that using low-rank policy adaptation with our approach achieved comparable performances to using full adaptation across all tasks and disturbance conditions. This demonstrates that LoRA can indeed be effectively combined with the differentiable simulation framework to achieve both sample- and parameter-efficient policy adaptation, where the latter may be particularly advantageous for fine-tuning pretrained policy networks that are significantly larger than the MLP used in our experiments.

2) Computational and Sample Efficiency Analysis: We analyze and compare the sample and computational efficiency of our approach against DATT for state-based tasks. For our approach, policy pretraining uses 300 epochs across 100 environments, which is equivalent to 4.5 million simulation steps in total, and takes approximately *15 seconds*. Empirically, we observed that good-performing initial policies can in fact be obtained using fewer epochs and environments, thanks to the low-variance first-order policy gradients from differentiable simulation. For online adaptation, each residual dynamics learning step (100 iterations) takes approximately *2 seconds*, and each policy adaptation step runs for 30 epochs across 10 environments (or 45k simulation steps) and takes about *1.5 seconds*. Here, with only 3 adaptation steps, which is equivalent to **4.5 seconds** of policy training in wall time, we already observe significant performance improvements for both hovering and tracking. In comparison, DATT trains the policy for 20 million simulation 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 [38], possibly exacerbating the high variance in the policy gradient estimates. In summary, 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.

TABLE III: 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 in green and orange.

Method	No Dist.	Small Dist.	Large Dist.
Base DiffSim (Vision)	0.133 ± 0.009	0.404 ± 0.039	1.383 ± 0.176
Ours (State)	0.015 ± 0.001	0.008 ± 0.002	0.105 ± 0.007
Ours (Vision)	0.082 ± 0.009	0.099 ± 0.021	0.207 ± 0.041
Ours (Vision, LoRA)	0.084 ± 0.014	0.111 ± 0.024	0.205 ± 0.048

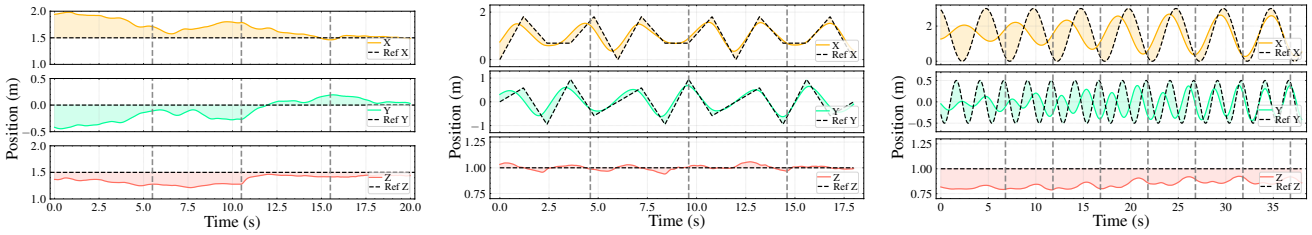


(a) Training time

(b) Final steady-state error

Fig. 6: (left) Policy training times using four different simulation configurations. (right) Resulting policy performances compared against the base policy performance. Error bars show ±3 standard deviations of the error distribution across 8 rollouts for each configuration.

3) Continuous Adaptation to Time-Varying Disturbances: We demonstrate the ability of our method to continuously adapt policies to unknown time-varying disturbances, using the realistic quadrotor simulator [12]. Here, we show an example of continuously adapting a state-based hovering policy under uniform, time-varying acceleration disturbance given by $\pm[0.5, 0.5, 0.5]\,\mathrm{m/s^2}$, which reverses its direction every 15 s. As shown in Fig. 5b, our approach rapidly adapts the state-based policy within 2 adaptation steps to adjust for each disturbance change, with the policy behavior remaining stable throughout the entire process. We observed that the continuous adaptation of a visual feature-based hovering policy was less stable than its state-based counterpart, consistent with our previous findings. For continuous trajectory tracking adaptation experiments, we provide detailed visualizations in the supplementary material.



(a) State-based hover with added mass and wind. (b) *5-Point Star* tracking with added wind. (c) *Figure-8* tracking on the large quadrotor.

Fig. 7: Rapid policy adaptation using our proposed approach in the real world. Vertical dashed lines indicate policy update steps which occur every 5 s. Shaded regions represent the error from the reference.

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 [12] and added a constant uniform acceleration disturbance of $2\,\mathrm{m/s^2}$ in the positive x-axis direction. We first collected 50 3-second rollout trajectories of the base hovering policy from random starting positions around the 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.

We first compared using the low-fidelity (low-fid) dynamics model (1) against using a high-fidelity model (high-fid), which simulates body and rotor drag effects, rotor thrust maps, and low-level controller dynamics, as the analytical dynamics model together with the residual dynamics network for forward simulation (res-fwd). 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-back) increases training time by approximately 30% without providing clear benefits to the policy performance. This is consistent with previous findings [20], [11] that combining accurate forward simulation with a surrogate gradient that points in approximately the same direction as the true gradient vector accelerates policy training without impacting the resulting policy performance.

D. Real World Validation

We conducted real-world experiments using the same tasks as the simulated experiments and the same pretraining and online adaptation procedures. A motion-capture system provides quadrotor state estimation at 100 Hz to the off-board workstation, which computes and sends commands to the on-board controller at 50 Hz. We used two quadrotors adapted from the Agilicious platform [12]: a small, lightweight quadrotor and a larger, heavier one with different dynamical properties (see Tab. IV). Moreover, we modified the small quadrotor by rigidly attaching to it a quadrotor stand from below, increasing its mass from 190 g to 260 g by approximately 37% and altering its inertial properties. Finally, we used a fan to create wind disturbances, resulting in complex, state-dependent forces on the modified quadrotor due to its highly imbalanced and non-uniform drag profile. Both the existing sim-to-real gap and the extra disturbances contribute to significant out-of-distribution dynamics that were unseen during policy pretraining.

TABLE IV: Quadrotor parameters for simulation and real-world experiments.

Param.	**Small Quadrotor**	**Large Quadrotor**
Mass [kg]	0.19	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]
Motor Time Constant [s]	0.025	0.033

Fig. 7a shows state-based hovering adaptation on the modified small quadrotor under a diagonal wind disturbance. Despite the more complex and unstable real-world disturbance forces compared to the constant uniform disturbance in simulation, our method still enables the policy to rapidly adapt with 2-3 policy update steps to compensate for disturbances. Similar results were observed for visual feature-based hovering, where the adaptation process appeared less stable than state-based hovering, which is consistent with our findings from simulated experiments. Real-world experiments also show that our approach achieves accurate trajectory tracking under added mass, wind, and significant model mismatches. In particular, our method achieves accurate tracking of the non-smooth *5-Point Star* under complex wind disturbances (see Fig. 7b). Moreover, Fig. 7c shows one particular experiment where a *Figure-8*-tracking policy was deployed on the large quadrotor. 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. Videos and visualizations of real-world experiments are provided in the supplementary material.

V. Conclusion

We propose a novel rapid policy adaptation framework 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 visual feature-based policies to unknown disturbances within *seconds*. One limitation of our framework lies in 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 dynamics. 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 aapplications," *IEEE Access*, 2021.

[2] E. Aljalbout, J. Xing, A. Romero, I. Akinola, C. R. Garrett, E. Heiden, A. Gupta, T. Hermans, Y. Narang, D. Fox, *et al.*, "The reality gap in robotics: Challenges, solutions, and best practices," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 9, 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 Int. Conf. Intell. Robots Syst.*, pp. 23–30, IEEE, 2017.

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

[5] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, pp. 982–987, Aug 2023.

[6] 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*, vol. 4, no. 26, p. eaau5872, 2019.

[7] 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.

[8] 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 Int. Conf. Robot. Autom.*, IEEE, 2024.

[9] H. Wang, J. Xing, N. Messikommer, and D. Scaramuzza, "Environment as policy: Learning to race in unseen tracks," in *2025 IEEE Int. Conf. Robot. Autom.*, pp. 11333–11339, IEEE, 2025.

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

[11] Y. Song, S. bae Kim, and D. Scaramuzza, "Learning quadruped locomotion using differentiable simulation," in *8th Conf. Robot. Learn.*, 2024.

[12] 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*, vol. 7, no. 67, p. eabl6259, 2022.

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

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

[15] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, "Neuralsim: Augmenting differentiable simulators with neural networks," in *2021 IEEE Int. Conf. Robot. Autom.*, IEEE, 2021.

[16] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza, "Data-driven mpc for quadrotors," *IEEE Robotics and Automation Letters*, 2021.

[17] 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.

[18] J. Eschmann, D. Albani, and G. Loianno, "Learning to fly in seconds," *IEEE Robotics and Automation Letters*, 2024.

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

[20] J. Heeg, Y. Song, and D. Scaramuzza, "Learning quadrotor control from visual features using differentiable simulation," in *2025 Int. Conf. Robot. Autom.*, IEEE, 2025.

[21] 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*, 2022.

[22] 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.

[23] 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*, 2023.

[24] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin, "Accelerated policy learning with parallel differentiable simulation," in *ICLR*, 2022.

[25] 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*, pp. 15418–15437, 2024.

[26] 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 Int. Conf. Robot. Autom.*, IEEE, 2025.

[27] 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 Int. Conf. Robot. Autom.*, pp. 9784–9790, IEEE, 2019.

[28] 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*, vol. 8, no. 1, pp. 312–319, 2022.

[29] 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*, vol. 7, no. 66, 2022.

[30] 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.

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

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

[33] L. Bo, T. Zhang, H. Zhang, J. Hong, M. Liu, C. Zhang, and B. Liu, "3d uav path planning in unknown environment: A transfer reinforcement learning method based on low-rank adaption," *Advanced Engineering Informatics*, vol. 62, p. 102920, 2024.

[34] J. Xing, A. Romero, L. Bauersfeld, and D. Scaramuzza, "Bootstrapping reinforcement learning with imitation for vision-based agile flight," in *8th Conf. Robot. Learn.*, 2024.

[35] 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.

[36] B. Zhang, A. Kapoor, and M. Sun, "Low-rank agent-specific adaptation (lorasa) for multi-agent policy learning," *arXiv preprint arXiv:2502.05573*, 2025.

[37] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, *et al.*, "Lora: Low-rank adaptation of large language models.," *ICLR*, vol. 1, no. 2, p. 3, 2022.

[38] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain generalization: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4396–4415, 2022.