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 fast, safe, and cost-effective development, but transferring them to the real world remains challenging due to the sim-to-real gap. Existing methods such as domain randomization and Real2Sim2Real improve robustness but either fail under out-of-distribution conditions or require costly retraining. In this work, we instead focus on rapid online adaptation. We propose a framework that unifies residual dynamics learning with real-time policy adaptation in a differentiable simulation. Starting from a simple dynamics model, the system continuously refines dynamics using real-world data to capture unmodeled effects such as payload changes and wind, and uses the refined dynamics to perform gradient-based, sample-efficient policy updates which are beyond reach for classical RL methods like PPO. Designed for speed, our approach adapts to unseen disturbances within *5 seconds* of training. We validate it on agile quadrotor control in simulation and the real world, achieving up to 81% lower hovering error than $\mathcal{L}_1$-MPC and 55% lower than DATT, while also demonstrating robustness in vision-based control without explicit state estimation.**

Supplementary Materials

Video: https://youtu.be/euK2GbcNTvk

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

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

I. Introduction & Related Works

Robot learning through simulation has advanced significantly with improved hardware and physics engines [1], offering a fast, safe, and cost-effective way to train policies [2]. However, transferring these policies to real systems remains challenging due to unknown parameters and unmodeled effects such as turbulence, sensor noise, and actuator delays [3]. This mismatch, known as the sim-to-real gap [4], continues to hinder reliable real-world deployment. Bridging this gap is crucial to preserve the benefits of simulation while ensuring robust performance under real-world variability. Domain randomization is a common strategy to address this issue [5], in which simulation parameters such as dynamics [6] and sensor noise [7] are varied during training to expose the policy to a wide range of possible deployment scenarios, however it still suffers from out-of-distribution failures [8]. Beyond domain randomization, Real2Sim2Real methods [9] have shown strong sim-to-real transfer capabilities, but require extensive data and costly offline retraining, making them unsuitable for rapid online adaptation to changing conditions [10].

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

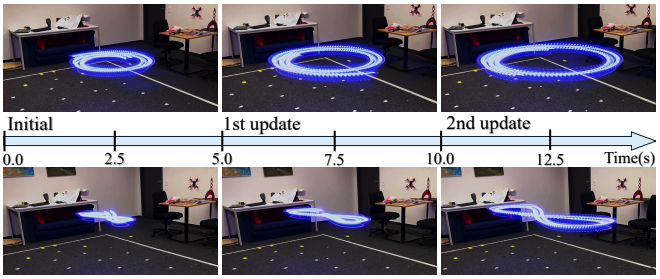


Fig. 1: Real-world trajectory tracking adaptation using our proposed approach. The policy rapidly learns within 2 updates (10 s of flight) to compensate for the large sim-to-real dynamics gap.

In this work, we approach these problems from a new 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 dynamics learning with rapid policy adaptation via differentiable simulation. We start with a lightweight point-mass 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 more accurate and sample-efficient policy adaptation. These processes are interleaved such that each batch of real-world data is used efficiently for both dynamics refinement and control improvement using the refined dynamics in simulation, where the gradient flow through the dynamics enables more efficient policy learning than classical RL approaches such as PPO. All components are designed to update both the residual dynamics and policy *as quickly as possible*. In this way, the policy becomes adaptive by continuously "overfitting" rapidly to the current environment scenario, and paradoxically, more "generalizable" across diverse conditions.

II. 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 point-mass dynamics model (see Appendix V-A) *without* residual dynamics. During online adaptation (see Fig. 2), 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 deployment loop uses

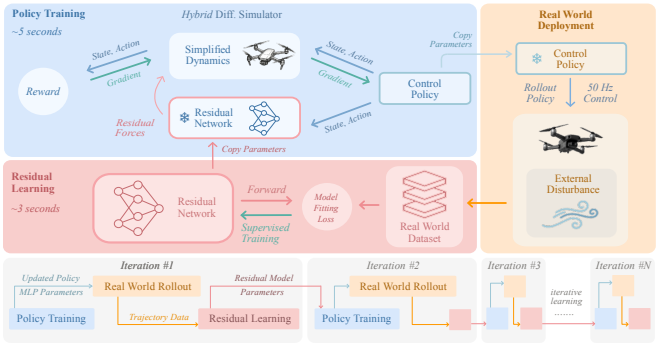


Fig. 2: Detailed illustration of the information flow within and between the three interleaved components of the proposed framework, including residual dynamics learning, differentiable simulation, and policy adaptation. These components operate concurrently across multiple threads in separate ROS nodes.

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 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. 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

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

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). The gradient and the update rule of the policy parameters ϕ are given by

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

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 open-source differentiable simulator for quadrotors [11] written in JAX to leverage both its automatic-differentiation framework for computing the analytical policy gradients and performing GPU-accelerated parallel simulation.

C. Residual Dynamics Learning

Given the concatenated input vector $[\boldsymbol{x}^\top, \boldsymbol{u}^\top] \in \mathbb{R}^{19}$ of quadrotor state $\boldsymbol{x}^\top = [\boldsymbol{p}^\top, \text{vec}(\boldsymbol{R})^\top, \boldsymbol{v}^\top] \in \mathbb{R}^{15}$ 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 $f_{\text{a}}(\boldsymbol{x}, \boldsymbol{u})$. 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 $\{[\boldsymbol{x}^\top, \boldsymbol{u}^\top]^i, \boldsymbol{a}_{\text{res}}^i\}_{i \in \mathcal{B}}$, we train the model by minimizing the loss function $\mathcal{L}_{\text{res}}$ via $\min_\theta \mathcal{L}_{\text{res}} = \min_\theta \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} ||\boldsymbol{a}_{\text{res}}^i - f_{\text{res}}([\boldsymbol{x}^\top, \boldsymbol{u}^\top]^i; \boldsymbol{\theta})||^2 + \beta \sum_{l=1}^{L} ||W^l||_2^2$, where W^l is the weight matrix of the l-th network layer.

D. 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 low-fidelity, analytical dynamics model 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. Consistent with prior work [11], we observed that combining accurate forward dynamics simulation with the backpropagation of a surrogate gradient based on a simplified dynamics model achieves faster runtime without impacting policy performance. We analyze and justify the above design choices through simulated experiments, and present and discuss the results in Appendix V-B.

III. 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. Trajectory tracking requires following a reference trajectory defined as a time-parameterized sequence of quadrotor states, with

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.

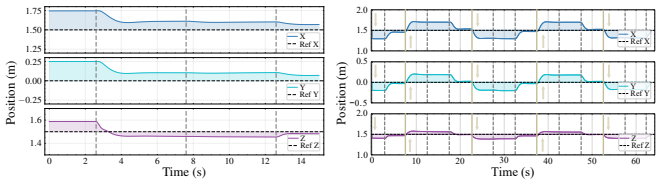
Method	No Dist.	Small Dist.	Large Dist.
Base DiffSim	0.128 ± 0.004	0.328 ± 0.001	1.228 ± 0.073
$\mathcal{L}_1$-MPC	0.091 ± 0.052	0.134 ± 0.073	0.552 ± 0.130
DATT (PPO)	0.013 ± 0.004	0.009 ± 0.005	0.231 ± 0.004
Ours	0.015 ± 0.001	0.008 ± 0.002	0.105 ± 0.007
Base DiffSim (Vision)	0.133 ± 0.009	0.404 ± 0.039	1.383 ± 0.176
Ours (Vision)	0.082 ± 0.009	0.099 ± 0.021	0.207 ± 0.041

policy training done in a state-based setting. 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 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 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 (150 simulation steps). For visual feature-based hovering, we use the initialization method in [11], and 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. 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: We compare against a state-of-the-art learning-based adaptive control method, Deep Adaptive Tracking Control (DATT) [12], which uses PPO with domain randomization and online $\mathcal{L}_1$ adaptive control-based disturbance estimation. We used the open-source implementation of [12] 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 +/- $[1, 1, 1]\,\text{m/s}^2$. We also compare against an adaptive Nonlinear MPC controller ($\mathcal{L}_1$-MPC) as implemented in [12], 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.



(a) Constant *large* disturbance. (b) Continuous *varying* disturbances.

Fig. 3: 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.

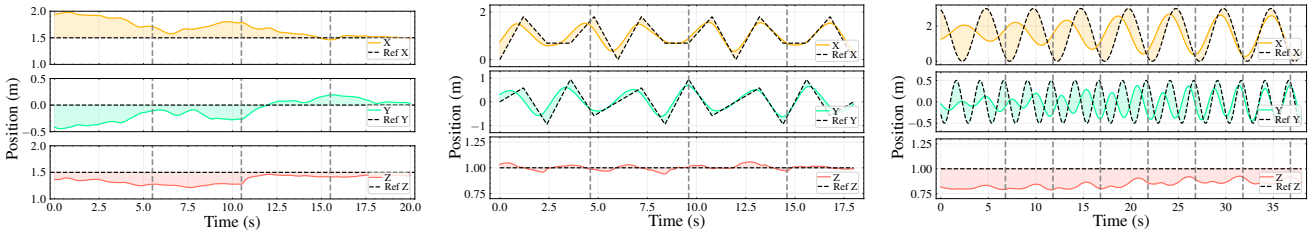
TABLE II: Average tracking errors (in m) for three different trajectories (*Circle*, *Figure-8*, and *5-Point Star*). The errors of the two best-performing methods for each disturbance condition are highlighted in green and orange.

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
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
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

B. Experimental Results

We used a realistic quadrotor simulator [13] equipped with the BEM model for aerodynamic effects and high-frequency simulation of controller dynamics. We simulated three levels of constant, uniform acceleration disturbances: $[0, 0, 0]\,\text{m/s}^2$ (*none*), $[0.5, 0.5, 0.5]\,\text{m/s}^2$ (*small*), and $[2, 2, 2]\,\text{m/s}^2$ (*large*). All experiments were run using an Nvidia RTX 4090 GPU (24 GB VRAM) with an Intel 14900KF CPU.

1) Performance Comparison to Baselines: For state-based stabilizing hover, we ran each method under all disturbance conditions from 8 different starting positions around the target, and used the final steady-state error as the performance metric. As summarized in Tab. I, 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. DATT performs well under both *none* and *small* disturbances which are within its training distribution, but struggles to adapt to the larger, out-of-distribution disturbance. Our visual feature-based approach resulted in larger errors than our state-based approach with less stable adaptation, which is likely due to partial state observability and sample inefficiency in learning vision-based control. We further demonstrate continuous adaptation to time-varying disturbances (see Fig. 3b),



(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. 4: 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.

where under periodically reversing acceleration disturbances ($\pm[0.5, 0.5, 0.5], \mathrm{m/s^2}$ every 15s), our method adapts within 2 steps after each change, maintaining stable performance throughout.

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, and exhibits consistent responsiveness to both smooth and non-smooth references. 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.

2) Computational and Sample Efficiency Analysis: We compare the sample and computational efficiency of our method against DATT on state-based tasks. Our policy pretraining requires 4.5M simulation steps (15 seconds), and in practice can be further reduced thanks to low-variance first-order gradients. For online adaptation, each residual update takes 2 seconds and each policy update 1.5 seconds, and with only 3 adaptation steps (4.5 seconds total), we already achieve significant performance gains. In contrast, DATT requires 20M simulation steps (2 hours) and exhibits slower convergence under larger domain randomization, likely due to the performance–generalization trade-off [14] and higher gradient variance. Overall, our approach improves compute and sample efficiency by simplifying pretraining and enabling fast online adaptation to out-of-distribution conditions.

C. 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 [13]: a small, lightweight quadrotor and a larger, heavier one with different dynamical properties (see Appendix V-C). 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.

Fig. 4a 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. 4b). Moreover, Fig. 4c 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.

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

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

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

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

[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*, pp. 326–340, PMLR, 2023.

[13] 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.

[14] 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.

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

V. Appendix

A. Low-Fidelity Quadrotor Dynamics Model

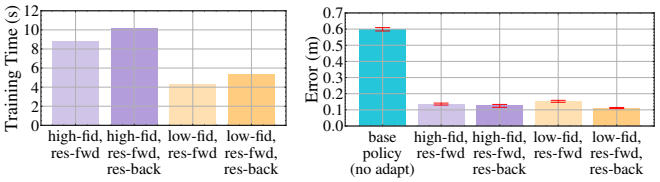
Given the quadrotor state $\boldsymbol{x}$ consisting of position $\boldsymbol{p} \in \mathbb{R}^3$, rotation matrix $\boldsymbol{R} \in \mathrm{SO}(3)$, and linear velocity $\boldsymbol{v} \in \mathbb{R}^3$, and commands $\boldsymbol{u}$ consisting of the mass-normalized collective thrust $c \in \mathbb{R}$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

$$\dot{\boldsymbol{x}} = \frac{d}{dt}\begin{bmatrix} \boldsymbol{p} \\ \mathrm{vec}(\boldsymbol{R}) \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \mathrm{vec}(\boldsymbol{R}[\boldsymbol{\omega}_{\text{cmd}}]_\times) \\ \boldsymbol{R}\boldsymbol{c} + \boldsymbol{g} \end{bmatrix} := f_{\text{a}}(\boldsymbol{x}, \boldsymbol{u}), \quad (3)$$

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. 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 [13] 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.



(a) Training time (b) Final steady-state error

Fig. 5: (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.

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, 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. 5a) than using the high-fidelity model, while the achieved policy performances by both methods were very comparable (see Fig. 5b). 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 [11], [15] 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.

C. Quadrotor Parameters

TABLE III: 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